高等学校计算机应用规划教材

软件测试技术与实践

- 软件与软件危机
- 软件测试基础
- 软件测试分类与分级
- 软件缺陷管理
- 基于生命周期的软件测试方法
- 软件测试过程及测试过程管理
- 软件静态测试
- 软件动态测试
- 软件单元测试
- 软件集成测试和确认测试
- 软件系统测试
- 面向对象软件测试



蔡建平　叶东升　康　妍
王爱菲　周百顺　李大奎　编著

清华大学出版社

高等学校计算机应用规划教材

软件测试技术与实践

蔡建平 叶东升 康 妍
王爱菲 周百顺 李大奎 编著

清华大学出版社

北 京

内 容 简 介

本书共分为软件测试基础、软件测试管理、软件测试方法与技术三部分，覆盖了软件测评的各个环节和知识点，内容包括软件及软件测试的基本概念、软件测试分类与分级、软件缺陷管理、软件全生命周期测试、软件测试及其过程管理、软件静态测试与动态测试，以及面向对象软件测试的方法等。对于其中的一些重要环节，设计了基于案例驱动，利用典型开源工具进行软件测试实践的教学内容，如缺陷管理、测试管理、静态测试、单元测试、集成测试、系统测试(包括功能测试及性能测试)等。

本书可作为高等院校计算机相关专业的教材和参考书籍，还可作为软件测试应用型人才的培训教材，也可供软件测试、软件质量保证及软件开发和软件项目管理从业人员参考。

图书在版编目(CIP)数据

软件测试技术与实践 / 蔡建平 等编著. —北京：清华大学出版社，2018
(高等学校计算机应用规划教材)
ISBN 978-7-302-48688-6

Ⅰ. ①软… Ⅱ. ①蔡… Ⅲ. ①软件－测试－高等学校－教材 Ⅳ.①TP311.5

中国版本图书馆 CIP 数据核字(2017)第 271394 号

责任编辑：王　军　李维杰
装帧设计：孔祥峰
责任校对：牛艳敏
责任印制：

出版发行：清华大学出版社
　　网　　址：http://www.tup.com.cn，http://www.wqbook.com
　　地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
　　社 总 机：010-62770175　　**邮　　购：**010-62786544
　　投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
　　质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**29　　**字　　数：**706 千字
版　　次：2018 年 1 月第 1 版　　**印　　次：**2018 年 1 月第 1 次印刷
印　　数：1~3000
定　　价：79.80 元

产品编号：

序

当前及今后，“软件定义一切”已充分说明了软件的重要性。软件作为产品及产品中的核心，其质量仍然可以认为是产品的生命。

在国家大力推进信息化与工业化融合之际，信息软件及工业软件的质量保证决定了“两化融合”的成败。软件质量保证的最重要手段之一就是软件测试。《软件测试技术与实践》作为航天中认推出支持两化软件测试的系列教材(《软件测试技术与实践》、《信息软件系统测试与实践》及《嵌入式软件测试与实践》)的基础，就显得非常重要。软件测试集技术、工程及实践于一身，如果没有好的技术基础和工程意识，眼高手低，那么在开展信息软件测试或工业软件测试时就会力不从心，多走弯路，无法把好质量关。

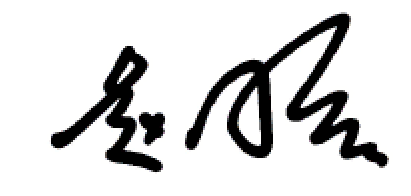
蔡建平教授长年从事软件工程、软件测试以及软件质量保证的研究、实践和教学，并为编写此书做了较长时间的准备，也出版了多本这方面的教材。特别是《软件测试方法与技术》被评为“全国工程硕士专业学位教育指导委员会推荐教材”和“软件工程专业核心课程系列教材”。本教材是在《软件测试方法与技术》和《软件测试实践教程》基础上改编而成的，具有如下主要特点：

(1) 该教材重要知识点的组织和讲述满足国内企业，特别是国内各种评测机构或组织对现代软件测试人才培养的要求；

(2) 该教材在传统软件测试技术和方法的基础上，特别强调软件测试是质量把控的重要手段之一，必须要与软件质量度量和评价相结合，要满足软件工程全生命周期软件测试的要求，要充分重视软件开发方法和应用方式对软件测试的影响，要注意软件测试工具对软件测试支持的重要作用等；

(3) 该教材还为高等院校和企业培训提供了软件测试课程实践教学方案、平台和案例，满足应用型软件测试人才培养的需要，满足软件测试工作人员的自学需要。

总之，该教材在软件测试技术的学习、普及、推广和软件测试应用型人才培养以及软件测试理论教学知识体系及实践教学体系的建立等方面进行了有益的探索和尝试。该教材内容全面、详实，实用性强。该教材的改版，将有益于国内软件测试人员和计算机相关专业的本科生及研究生的学习与能力的培养，有益于推动现代软件测试技术和方法的研究、教学和实践的进一步发展，同时对我国软件测试业的发展和软件测试紧缺人才的培养起到积极的促进作用。



航天中认软件测评科技(北京)有限责任公司总经理

2017 年 8 月 30 日于北京

前　　言

当前，软件测试已从传统的软件工程瀑布模型中测试阶段的软件测试变化为覆盖包括需求分析、系统设计、详细设计、程序编码、内部测试、系统测试、系统安装、确认验收以及系统维护整个软件工程生命周期的软件测试；从过去单纯的测试概念发展到包括静态分析、质量度量与评价在内的评测结合的软件评测思想；从传统的测试内容分类到基于质量特性、子特性的测试内容分类；从传统的结构化程序测试方法到面向对象的软件测试方法；从早期的单机或桌面测试到网络应用测试及嵌入式应用测试；从以手工测试为主发展到离不开测试工具支持的测试及管理。事实上，软件测试也成为耗费人力、财力和时间的一项复杂的工作，对测试人员提出了高素质、专业化的要求。对软件测试人员不但要求精通各种软件测试技术和方法，有一定的软件测试工程实践经验；还要求他们熟悉软件开发技术和软件开发流程，具有快速学习专业知识或领域知识、掌握新技术和应用新工具的能力；另外，软件测试人员要有团队合作意识，善于和人沟通与交流，并能承受被人误解和指责的心理素质。

随着计算机技术的快速发展，软件越来越普遍地应用到各个领域和各个方面，且应用规模越来越大，应用形式越来越复杂，软件质量要求越来越高，软件测试越来越重要。对于高等院校而言，人才的培养是其核心的工作，鉴于高素质的软件测试专业人才越来越奇缺，航天中认积极开展校企合作，与大连理工大学软件学院共建软件测试课程和实验室，大力开展教学实践、教学改革的探索和实践，编写专业教材，凝练软件测试人才培养的成果。

本书是在《软件测试方法与技术》和《软件测试实践教程》基础上改编而成，将两本书的内容压缩、整合和完善，并试图将本书重点突出在两方面：软件测试技术与软件测试实践。本教材既包括成熟的理论基础，也包括软件测试工具的使用，同时还可以通过书中案例实践的学习巩固学习成果。本书是作者多年从事软件测试技术研究、软件测试课程教学及软件项目测试成果和经验的总结。全书共分 12 章，分为 3 部分：第 I 部分(第 1 至第 3 章)是软件测试基础，涉及软件测试的一些基本概念和基础知识，如软件与软件危机、软件测试基本概念、软件测试分类与分级；第 II 部分(第 4 至第 6 章)是本书的重点内容之一，详细讲述生命周期的软件测试方法与技术，包括软件缺陷与缺陷管理、软件测试及其过程管理、软件测试管理工具的使用及案例实践；第III部分(第 7 至第 12 章)也是本书的重点内容之一，详细讲述软件测试的方法与技术，包括软件静态测试及软件动态测试。其中，基于 McCabe 设计与集成复杂性的集成测试方法解决了集成覆盖测试的理论问题。

本书系统全面地从软件测试基础理论到软件测试全生命周期实践角度，系统地为读者解决从事软件测试工作的问题。另外，本书几乎在各章对支撑该章软件测试方法和技术应用的开源软件测试工具详细地进行了应用介绍，并配有具体测试案例。这些工具及案例的

应用对于支持高校软件测试课程实践是有意义的。最后，本书取材新颖、内容翔实、通俗易懂、技术实用、覆盖面广、指导性强，既可作为软件测试相关课程的研究生(特别是工程硕士专业学位研究生)与本科生的教材，同时还可供软件测试培训和软件测试人员自学的书籍。

致谢，再一次感谢《软件测试方法与技术》和《软件测试实践教程》中提到的人员，感谢取材互联网上的有关原创作者，感谢清华大学出版社的大力支持和帮助。

蔡建平

2017 年 8 月 28 日于北京

作 者 简 介



蔡建平，在军队从事教学与全军军用共性软件、软件工程、软件质量保证等项目的论证及研究工作 20 多年，获军队科技进步一等奖一项、二等奖两项、三等奖两项，编著《Ada 程序设计语言高级教程》，发表各类学术文章 20 多篇。

在企业工作期间，主持开发了嵌入式软件工程和软件测试工具，这些工具已成功地用于航空、航天等国防项目的测试和软件工程化，极大地保证了这些项目的质量。

在北京工业大学工作期间，在软件学院的学科、专业、实验室、“211 工程”、教育部和北京市特色专业、科技创新平台以及学科交叉等建设方面做了大量的工作，取得了突出成果。获国家教育教学成果二等奖。“软件测试”及“高级软件编程技术”分别评为学校精品课程和研究生重点建设课程，《软件测试大学教程》、《软件测试实验指导教程》、《嵌入式软件测试实用技术》、《软件综合开发案例教程》4 部教材和专著已在清华大学出版社出版发行。其中《软件测试大学教程》于 2013 年被评为全国工程硕士专业学位教育指导委员会推荐教材。

科研上，发表各类论文 20 多篇，申请专利、软著多项，指导的学生科技活动成果获第十二届“挑战杯”全国大学生课外学术科技作品竞赛三等奖，指导的两篇硕士论文被评为校优秀论文。

作为惠普国际软件人才及产业基地的学术总监，负责全国各高校共建专业合作论证及顶层策划与设计，培养方案、课程体系、实训方案的设计与制定，与实训课程配套教材的研发组织及各门课程的研发组织，基地师资队伍及课程团队建设，以及 1000 多名学生实习/实训的组织与实施。

目前，在航天中认负责公司的咨询、研究及对内和对外的技术培训等业务，负责和参与了交通部、体育总局、中海油、大连理工大学、浪潮、长虹、美的、小天鹅、格力、轨道交通、国家电网、中国质量认证中心、汽车电子、医疗电子、家用电器等信息化建设项目及嵌入式系统项目的软件工程化、软件质量保证、软件测试以及配套实验室建设的咨询与培训。

蔡建平教授还是国家科学技术奖励、国家专利奖励、山东省科学技术奖励、北京市科学技术奖励、海淀区科学技术奖励、北京市文化创意产业、海淀区文化创意产业等专家库成员。

目　　录

第 I 部分　软件测试基础篇

1947 年，计算机还是由机械式继电器和真空管驱动的、有房间那么大的庞然大物，由哈佛大学制造的 MARK-II 则是体现当时技术水平的计算机。在一次整机运行中，它突然停止了工作。技术人员爬到计算机上找原因，发现是一只飞蛾受光和热的吸引飞到了计算机内部一组继电器的触点之间，然后被高电压击死。由此，计算机的缺陷产生了，虽然最后该缺陷被技术人员解决了，但是我们从此认识到了它就是缺陷。

如今软件已经渗透到我们的日常生活中，从办公设备到家用电器，从通信工具到航空航天事业，软件无处不在，然而却又很难完美无缺。

1994 年的秋天，迪士尼公司发布了第一个面向儿童的多媒体光盘——《狮子王动画故事书》(*The Lion King Animated Storybook*)。对此，迪士尼公司做了大量的宣传。因此，销售额非常可观。然而圣诞节过后，公司接到了大量的投诉电话，称游戏不能运行。经证实，造成这种后果的原因是迪士尼公司未对市面上使用的许多不同类型的 PC 机型进行测试，软件只能在少数系统中运行。

同样是 1994 年，英特尔奔腾浮点除法缺陷事件，不仅使英特尔公司的形象受到严重影响，并且为自己处理软件缺陷的行为付出了 4 亿多美元的代价。

类似的还有美国航天局火星极地登陆者号探测器事件、爱国者导弹防御系统事件、千年虫问题等，这些事件的后果有的是带来不便，例如游戏玩不成，有的可能是灾难性的后果——导致机毁人亡。它们的发生都是由于在软件中隐藏着错误。

软件为什么会频繁出问题，如何杜绝或将它们减至最少，将影响降至最低呢？在论述软件测试概念之前先介绍一下软件、软件危机及软件工程等概念，然后再讲解软件测试的相关知识。

第1章　软件与软件危机

我们都知道软件的重要意义：软件是信息化的核心，现代国民经济、国防建设、社会发展及人民生活都离不开软件。软件产业是增长最快的朝阳产业，是高投入、高产出、无污染、低能耗的绿色产业。软件产业关系到国家经济和文化安全，体现了国家综合实力，是决定未来国际竞争地位的战略性产业。

但软件到底什么？它具有什么样的特性？它能够干什么？

1.1.1　软件特性

软件是人通过智力劳动产生的，软件产品是人的思维结果，是一个逻辑部件，而不是一个物理部件。因此，软件生产水平最终在相当程度上取决于软件人员的教育、训练和经验的积累。所以，软件具有与硬件不同的一些特点。

1. 软件与硬件的不同

软件与硬件的不同或差别主要反映在以下几个方面。

1) 表现形式不同

硬件有形，有色，有味，看得见，摸得着，闻得到。而软件无形，无色，无味，看不见，摸不着，闻不到。软件大多存在于人们的脑海里或纸面上，它的正确与否，是好是坏，一直要到程序在机器上运行才能知道。这就给设计、生产和管理带来许多困难。

2) 生产方式不同

软件开发是智力的高度发挥，而不是传统意义上的硬件制造。尽管软件开发与硬件制造之间有许多共同点，但这两种活动是根本不同的。在两种活动中，通过好的设计能够得到好的质量，但硬件制造阶段可能引入的质量问题在软件开发中却不会出现，反之亦然。这两种活动都依靠人，但人的作用和工作专长之间的关系是完全不同的。因为软件是逻辑产品，如几个人共同完成一个软件项目时，人与人之间就有一个思想交流的问题，称之为通信关系。通信是要付出代价的，不仅要花费时间，现实中由于通信中的疏忽常常会使错误增加。人虽然是最聪明的，但人也是最容易犯错误的。

3) 要求不同

硬件产品允许有误差，如加工一根轴，其外径精度要求为Φ500±0.1。生产时，只要达到规定的精度要求就算合格。而软件产品却不允许有误差，要 1 就是 1。如美国金星探测器水手 1 号，导航程序的一条语句的语法正确，但语义错了，结果飞行偏离航线，最终导致试验的失败。又如，阿波罗宇宙飞船飞行控制软件，由于粗心，把一个逗号写成一个句号，又没能检查出来，几乎造成悲剧性的后果。这就给软件开发和维护，以及它的质量保证体系提出了很高的要求。

4) 维护不同

硬件是会用旧、用坏的这是因为硬件在使用过程中，由于受到环境的影响，如灰尘、温湿度变化、空气污染、振动等因素而使产品产生腐蚀或磨损，使硬件故障率增加，甚至损坏，以致不能使用。解决的办法，换上一个相同的备件就是了。而软件不受那些引起硬件损坏的环境因素的影响。因此，在理论上，软件不会用旧、用坏。但实际上，软件也会变旧、变坏。因为在软件的整个生命周期中，一直处于改变(维护)状态。而随着某些缺陷的改变，很可能引入一些新的缺陷，因而使软件的故障率增加，品质变坏。硬件某一部分变坏，可以使用备件，而软件则不存在这种备件，因为软件中任何缺陷都会在机器上导致同样的错误。所以，软件维护要比硬件维护复杂得多。

从上我们可以总结出软件具有同传统的工业产品相比的一些特性。

2. 软件的特性

1) 软件是一种逻辑实体，具有抽象性

这个特点使它与其他工程对象有着明显的差异。人们可以把它记录在纸上、内存中和磁盘、光盘上，但无法看到软件本身的形态，必须通过观察、分析、思考、判断，才能了解它的功能、性能等特性。

2) 软件没有明显的制造过程

一旦研制开发成功，就可以大量拷贝同一内容的副本。所以对软件的质量控制，必须着重在软件开发方面下工夫。

3) 软件在使用过程中，没有磨损、老化的问题，但有退化问题

软件在生命周期后期不会因为磨损而老化，但会为了适应硬件、环境以及需求的变化而进行修改，而这些修改又不可避免地会引入错误，导致软件失效率升高，从而使得软件退化。当修改的成本变得难以接受时，软件就被抛弃。

4) 软件对硬件和环境有着不同程度的依赖性

这导致软件移植的问题。

5) 软件的开发至今尚未完全摆脱手工作坊式的开发方式，生产效率低

6) 软件是复杂的，而且以后会更加复杂

软件是人类有史以来生产的复杂度最高的工业产品。软件涉及人类社会的各行各业、方方面面，软件开发常常涉及其他领域的专门知识，这对软件工程师提出了很高的要求。

7) 软件的成本相当昂贵

软件开发需要投入大量、高强度的脑力劳动，成本非常高，风险也大。现在软件的开销已大大超过了硬件的开销。

8) 软件工作牵涉很多社会因素

许多软件的开发和运行涉及机构、体制和管理方式等问题，还会涉及人们的观念和心理。这些人的因素，常常成为软件开发的困难所在，直接影响到项目的成败。

1.1.2　软件种类

软件已经渗透到我们的日常生活，用于各行各业。具体地说，软件按如下形式分类：

(1) 系统软件(如操作系统、数据库管理系统、设备驱动程序、通信处理程序等)；

(2) 应用软件(如事务软件、实时软件、工程和科学软件、嵌入式软件、娱乐软件、个人计算机软件、人工智能软件等)；

(3) 工具软件(如文本编辑软件、文件格式化软件、磁盘向磁带传输数据的软件、程序库系统以及支持需求分析、设计、实现、测试和管理的软件等)；

(4) 可重用软件。

1.2　软件危机

前面提到的美国于 20 世纪 60 年代初飞向金星的第一个空间探测器(水手 I 号)，因其飞行舱内计算机导航程序中一条语句的语义出错，致使偏离航线无法取得成功。这条语句的错误并不是语法错误，而是具有完全不同于程序员所期望的意思——语义错误。可以称得上是世界上最精心设计，并花费了巨额投资的美国阿波罗登月飞行计划的软件，仍然没有避免出错；另外，阿波罗 8 号太空飞船的一个计算机软件错误，造成存储器的一部分信息丢失；还有，阿波罗 14 号在飞行的 10 天里，出现了 18 个软件错误。当时，软件系统的可靠性得不到保证，几乎没有不存在错误的软件系统。

那时，计算机已有近 20 年的历史，一方面硬件成本每隔两到三年降低一半，内存和外存则成本每年降低 40%左右，硬件性能价格比每十年提高一个数量级，但所需的软件很少能在成本、时间进度、功能规模、维护能力等方面达到要求，特别是可靠性难以符合人们的需要。正如 E.E.David 指出的那样，大型系统的软件生产已经成为管理人员担惊受怕的项目，于是，人们在 20 世纪 60 年代后期惊呼发生了软件危机(software crisis)。

1.2.1　软件危机的分析

产生软件危机的原因是多方面的，但不管怎样，软件危机有它内在的或本质上的原因。下面就软件危机产生的诸多原因进行分析，揭示软件危机内在或本质上的原因。

1. 早期编程的特点

从 20 世纪 40 年代开始，人们从在 MARK-I 和 ENIAC 计算机上编制程序，到软件危机发生时为止的 20 多年时间里，对软件开发的理解就是编程，且编程是在一种无序的、崇尚个人技巧的状态中完成的。

和今天的软件开发相比，那时的编程具有一些特点。

1) 软件规模相对较小

原因有二：

① 人们对软件可能达到的功能认识有限，那时最为关心的是计算机硬件的发展。作为一名计算机专业人员，他不太关心软件问题(只有为数不多的专业人员才去关心软件)，

但他必须懂得计算机的结构。作为一个机构，其大量资金也被用于计算机硬件开销，软件只是作为展现其软件性能的一种手段而投入少量资金。

② 硬件性能从某种意义上左右着人们对软件的需求，人们总是不自觉地在心中盘算着硬件支持的可能性，这种现象从根本上阻碍了软件的广泛应用，从而限制了软件规模。

2) 编程作为一门技艺

大部分软件技术人员不太关心他人的工作，他们往往陶醉于自己的编程技巧，并且那时也无编程规范与标准，这也是由软件规模决定的。决定软件质量的唯一因素就是编程人员的素质，时间进度亦是如此。根据 H.Sackman 等人的调查，素质好的人与素质差的人，在软件生产效率上的比例是 10∶1，在程序处理速度和存储容量上的比例是 5∶1。

3) 缺少有效方法与软件工具的支持

在当时几乎谈不上有效的编程方法，使用最多的亦只是简单的控制流图。软件工具只有子程序库、装入程序、编辑程序、排错程序以及汇编和编译程序(后来才有链接程序)，以至于许多程序员沉溺于编程技艺的掌握和使用上。

4) 不重视软件开发的管理

由于人们重视个人技能，再加上软件开发过程能见度低，许多管理人员甚至根本不知道软件技术人员在干什么，究竟做得如何，从而造成管理活动几乎不存在。直到今天，这种观念在一些机构中仍有残留，为此，我们对软件开发的管理问题必须给予更多的重视。

5) 软件开发后的维护工作很难进行

由于人们重视个人技能，一旦需要做某些修改，就要原编程人员进行修改。如果他已离开本机构，就需要别人去读懂他的程序，再进行修改和补充。此项工作极其辛苦，说不定修改了一处，反而出现上百处漏洞，变得得不偿失。

上述编程特点导致出现人们常说的软件危机现象。

2. 大型软件开发问题

进入 20 世纪 60 年代，应客观需求需要制作一些大型软件，这样就出现了像第一个空间探测器所描述的例子。国外在开发一些大型软件系统时遇到了许多困难，有些系统最终彻底失败了；有些系统虽然完成了，但比原定计划推迟了好几年，而且费用大大超出了预算；有些系统未能达到用户当初的期望；有些系统则无法进行修改、维护。IBM 公司 OSS/360 系统和美国空军某后勤系统都花费了几千人/年努力，历尽艰辛，但结果令人失望。

随着软件开发应用范围的增广，软件开发规模越来越大。大型软件开发项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件开发系统的经验，而多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确，有时还会产生误解。软件开发项目开发人员不能有效地、独立自主地处理大型软件开发的全部关系和各个分支，因此容易产生疏漏和错误。这也是导致软件危机产生的一个原因。

3. 软件生产的知识密集和人力密集的特点

由于计算机技术和应用发展迅速，知识更新周期加快，软件开发人员经常处在变化之中，不仅需要适应硬件更新的变化，而且还要涉及日益扩大的应用领域问题研究；软件开

发人员所进行的每一项软件开发几乎都必须调整自身的知识结构以适应新的问题求解的需要，而这种调整是人所固有的学习行为，难以用工具代替。

软件生产的这种知识密集和人力密集的特点是造成软件危机的根源所在。从软件危机的种种表现和软件作为逻辑产品的特殊性，可以发现软件危机的具体原因。

4. 用户需求难以明确

对于大型软件往往需要许多人合作开发，甚至要求软件开发人员在软件开发过程中深入应用领域对相关问题进行研究，了解用户需求。这样就需要在用户与软件开发人员之间以及软件开发人员之间相互通信。在此过程中难免发生理解上的差异，导致用户需求不明确的问题产生。

用户需求不明确问题主要体现在四个方面：

① 在软件开发出来之前，用户自己也不清楚软件开发的具体需求；

② 用户对软件开发需求的描述不精确，可能有遗漏，有二义性，甚至有错误；

③ 在软件开发过程中，用户还提出修改软件开发功能、界面、支撑环境等方面的要求；

④ 软件开发人员对用户需求的理解与用户本来的愿望有差异。这些需求问题将导致后续软件错误的设计或实现，而要消除这些需求上的误解和错误往往需要付出巨大的代价。这同样是产生软件危机的一个原因。

5. 缺乏正确的理论指导，缺乏有力的方法学和工具方面的支持

由于软件开发不同于大多数其他工业产品，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性，因此加剧了软件开发产品的个性化，这也是发生软件开发危机的一个重要原因。

6. 软件开发复杂度越来越高

软件开发不仅仅是在规模上快速地发展扩大，而且其复杂性也急剧地增加。软件开发产品的特殊性和人类智力的局限性，导致人们无力处理复杂问题。所谓复杂问题的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又摆在人们的面前。

7. 软件危机的本质原因

从前面软件危机的原因分析来看，软件危机产生的本质原因主要有两点：

① 与软件本身的特点有关；

② 与软件的开发人员有关。

从上我们可以看出，软件危机是指在计算机的开发和维护过程中所遇到的一系列严重问题。这些问题绝不仅仅是不能正常运行的软件才具有的，实际上，几乎所有软件都不同程度地存在这些问题。

1.2.2　软件危机现象

事实上，软件危机包含着两方面的问题：

① 如何开发软件，以满足对软件日益增长的需求；

② 如何维护数量不断膨胀的已有软件。具体来说，软件危机主要有以下一些典型表现。

1. 对软件开发成本和进度的估计常常很不准确

实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

2. 用户对已完成的软件系统不满意的现象经常发生

软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就仓促上阵匆忙着手编写程序。软件开发人员和用户之间的信息交流往往很不充分，闭门造车必然导致最终的产品不符合用户的实际需要。

3. 软件产品的质量往往靠不住

软件可靠性和质量保证的确切定量概念刚刚出现不久，软件质量保证技术还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

4. 软件常常是不可维护的

很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。可重用的软件还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

5. 软件通常没有适当的文档资料

计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是最新式的(即和程序代码完全一致)。软件开发组织的管理人员可以使用这些文档资料作为里程碑，来管理和评价软件开发工程的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息；对于软件维护人员而言，这些文档资料更是至关重要、必不可少的。缺乏必要的文档资料或者文档资料不合格，必然给软件开发和维护带来许多严重的困难和问题。

6. 软件成本在计算机系统总成本中所占的比例逐年上升

由于微电子学技术的进步和生产自动化程度的不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在 1985 年软件成本大约已占计算机系统总成本的 90%。

7. 软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及深入的趋势

软件产品供不应求的现象使人类不能充分利用现代计算机硬件提供的巨大潜力。

以上列举的仅仅是软件危机的一些明显的表现，与软件开发和维护有关的问题远远不止这些。总之，软件危机一方面与软件本身的特点有关，另一方面与软件开发和维护的方法不正确有关。

1.2.3　避免软件危机的方法

如何避免软件危机的产生是一个非常大的课题，是否存在非常有效、十分管用的方法现在还未有结论。以下两点是在软件开发过程中为避免软件危机问题的出现通常要求大家要努力做到的。

1) 应该对计算机软件有一个正确的认识

应该彻底清除在计算机系统早期发展阶段形成的软件就是程序的错误观念，一个软件必须由一个完整的配置组成。事实上，软件是程序、数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构；文档是开发、使用和维护程序所需要的图文资料。1983 年 IEEE 为软件下的定义是：计算机程序、方法、规则、相关的文档资料以及在计算机上运行程序时所必需的数据。虽然表面上看，这个定义列出了软件的五个配置成分，但是，方法和规则通常是在文档中说明并在程序中实现的。

2) 必须充分认识到软件开发不是某个个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目

必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法，特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。极力推广和使用在实践中总结出来的开发软件的成功技术和方法，并且研究探索更好、更有效的技术和方法，使用更好的开发工具。这样，在软件开发中就会最大限度地消除软件危机的出现。

1.3　软件工程

1968 年秋季，NATO(北大西洋公约组织，简称北约)的科技委员会召集了近 50 名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱软件危机的对策。在那次会议上第一次提出了软件工程这个概念。到现在，软件工程走过了约 50 年的历程。

在这约 50 年的发展历程中，人们针对软件危机的表现和原因，经过不断的实践和总结，越来越认识到：按照工程化的原则和方法组织软件开发工作，是摆脱软件危机的一条主要出路。

1.3.1　软件工程定义

今天，尽管软件危机并未被彻底解决，但软件工程近 50 年的发展仍可以说是硕果累累。下面我们给出软件工程的定义，然后简单讨论一下软件工程所包括的内容。

软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法进行软件的开

发和维护的学科。它作为一门新兴的工程学科，主要研究软件生产的客观规律性，建立与系统化软件生产有关的概念、原则、方法、技术和工具，指导和支持软件系统的生产活动，以期达到降低软件生产成本、改进软件产品质量、提高软件生产率水平的目标。软件工程学从硬件工程和其他人类工程中吸收了许多成功的经验，明确提出了软件生命周期的模型，发展了许多软件开发与维护阶段适用的技术和方法，并应用于软件工程实践，取得了良好的效果。

1. 软件工程的具体含义

软件工程的具体含义体现在四个方面：

(1) 把软件开发看成一项有计划、分阶段、严格按照标准或规范进行的活动(软件工程是指导计算机软件开发和维护的工程学科，软件工程方法=工程方法+管理技术+技术方法)；

(2) 将系统的、规范的、可度量的方法应用于软件的开发、运行和维护的过程(将工程化应用于软件中，并研究提到的上述途径)；

(3) 要求采用适当的软件开发方法和支持环境及编程语言来表示和支持软件开发各阶段的各种活动，并使开发过程条令化、规范化，使软件产品标准化、开发人员专业化；

(4) 用工程学的观点进行费用估算，制定进度，制定计划；用管理科学中的方法和原理进行软件生产的管理；用数学的方法建立软件开发中的各种模型和各种算法。

2. 软件工程三要素

软件工程包括三个要素：方法、工具和过程。

1) 软件工程方法

软件工程方法为软件开发提供了如何做的技术。它包括了多方面的任务，如项目计划与估算，软件系统需求分析，数据结构、系统总体结构的设计，具体算法的设计、编码、测试以及维护等。

2) 软件工具

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，这些软件工具集成起来，建立起称之为计算机辅助软件工程(Computer Aided Software Engineering，CASE)的软件开发支撑系统。CASE 将各种软件工具、开发机器和存放开发过程信息的工程数据库组合起来，形成软件工程环境。

3) 软件工程过程

软件工程过程则是将软件工程方法和软件工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理及软件开发各个阶段完成的里程碑。

软件工程是一种层次化的技术。任何工程方法(包括软件工程)必须以有组织的质量保证为基础。全面的质量管理和类似的理念刺激了不断的过程改进，正是这种改进导致更加成熟的软件工程方法的不断出现。支持软件工程的根基就在于对质量的关注。

3. 软件工程基本原理

著名软件工程专家 B.W.Boehm 综合有关专家和学者的意见并总结多年来开发软件的经验，于 1983 年在一篇论文中提出了软件工程的七条基本原理。

1) 用分阶段的生命周期计划进行严格的管理

统计表明，不成功的软件项目中有 50%左右是由于计划不周造成的。应该把软件生命周期划分为若干阶段，并制定出相应的切实可行的计划，严格按照计划对开发和维护进行管理。B.W.Boehm 认为，应制定和严格执行 6 类计划：项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划、运行维护计划。

2) 坚持进行阶段评审

设计的错误占软件错误的 63%，编码错误只占 37%。而且在后期纠正错误的代价非常高。因此，必须严格坚持阶段评审，及早发现和纠正错误。

3) 实行严格的产品控制

在现实中，由于外部原因要求对需求等进行修改是难免的，但必须有严格的管理制度和措施。

4) 采用现代程序设计技术

如结构化程序分析、结构化程序设计和面向对象程序设计等。

5) 软件工程结果应能清楚地审查

由于软件是一种看不见、摸不着的逻辑产品，对它的检验和审查很困难。因此，应提供可视化的检验标准和方法。

6) 开发小组的人员应该少而精

软件开发小组的人员应该是素质高，人员不宜过多。人员素质低和人员过多，都会导致软件的错误率高，且开发效率下降，成本增加。

7) 承认不断改进软件工程实践的必要性

软件工程是一门不断迅速发展的学科，必须学习和跟踪先进的技术和方法，也要不断地总结经验、改进方法，要不断进行技术创新。

B.W.Boehm 指出，遵循前六条基本原理，能够实现软件的工程化生产；按照第七条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。

4. 软件工程框架

软件工程(Software Engineering)框架可概括为：目标、过程和原则。

1) 软件工程的目标

软件工程的目标是生产具有正确性、可用性、开销适宜并且软件项目成功的软件产品。正确性指软件产品达到预期功能的程度；可用性指软件基本结构、实现及文档为用户可用的程度；开销适宜是指软件开发、运行的整个开销满足用户要求的程度；软件项目成功指开发成本低、功能与性能满足需求、易于移植、维护方便以及按时完成开发任务并及时交付软件产品。这些目标的实现不论在理论上还是在实践中均存在很多待解决的问题，它们

形成了对过程、过程模型及工程方法选取的约束。

2) 软件工程的过程

软件工程的过程是指生产一个最终能满足需求且达到工程目标的软件产品所需要的步骤，主要包括开发过程、运作过程、维护过程。它们覆盖了需求、设计、实现、确认以及维护等活动。需求活动包括问题分析和需求分析。问题分析获取需求定义，又称软件需求规约；需求分析生成功能规约。设计活动一般包括概要设计和详细设计。概要设计建立整个软件系统结构，包括子系统、模块以及相关层次的说明、每一模块的接口定义；详细设计产生程序员可用的模块说明，包括每一模块中的数据结构说明及加工描述。实现活动把设计结果转换为可执行的程序代码。确认活动贯穿于整个开发过程，实现完成后的确认，保证最终产品满足用户的要求。维护活动包括使用过程中的扩充、修改与完善。伴随以上过程，还有管理过程、支持过程、培训过程等。

3) 软件工程的原则

软件工程的原则是指围绕工程设计、工程支持以及工程管理在软件开发过程中必须遵循的原则，具体为：

① 采取适宜的开发模型，用以控制易变的需求。

② 采用合适的设计方法，支持软件的模块化、抽象与信息隐藏、局部化、一致性以及适应性等设计要求。

③ 提供高质量的工程支持，特别要强调软件工具和环境对软件过程的支持。

④ 重视开发过程的管理，要有效利用可用的资源、生产满足目标的软件产品、提高软件组织的生产能力等。

6. 软件工程的本质特征

基于软件特性及软件危机产生的原因，我们可以清楚地了解软件工程的本质特征，即：

① 软件工程关注于大型程序的构造。

② 软件工程的中心课题是控制复杂性。

③ 软件经常变化(控制和管理)。

④ 开发软件的效率非常重要(工具与环境)。

⑤ 和谐地合作是开发软件的关键(团队精神)。

⑥ 软件必须有效地支持它的用户。

⑦ 在软件工程领域中是由一种文化背景的人为另一种文化背景的人创造产品。

7. 软件开发技术和软件项目管理

软件工程包括软件开发技术和软件项目管理两方面的内容。

1) 软件开发技术与开发方法

软件开发技术是为了完成软件生命周期各阶段的任务所必须具备的技术手段。软件开发技术包括：①软件开发方法(是一种使用早已定义好的技术集及符号表示习惯来组织软件生产过程的方法，一般表述成一系列的步骤，每一步都与相应的技术和符号相关，目的是在规定的投资和时间内开发出符合用户需求的高质量的软件)。②软件工具(是为了支持软

件人员开发和维护而使用的软件，它可以放大人类的智力、提高工作效率、便于管理实施，并为软件开发方法提供了自动的或半自动的软件支撑环境，辅助软件开发任务的完成。当前，在软件开发过程中人们越来越重视工具的使用，用以辅助进行软件项目管理与技术生产)。③软件环境(是将软件生命周期各阶段使用的软件工具有机地集合成为一个整体，形成能够连续支持软件开发与维护全过程的集成化软件支援环境，用以开发软件，提高开发效率和软件质量，降低开发成本，以期从管理和技术两方面解决软件危机问题)。

在软件开发过程中常用的软件开发方法有：

(1) 面向数据流的结构化程序开发方法(最终关注程序结构)。

- 指导思想：自顶向下，逐步求精。
- 基本原则：功能的分解与抽象。
- 适合于数据处理领域的问题。

(2) 面向数据结构的开发方法——Jackson 方法。

- JSP(Jackson Structured Programming)：首先描述问题的输入/输出数据结构，分析其对应性，然后推出相应的程序结构，从而给出问题的软件过程描述。以数据结构为驱动。
- JSD(Jackson Structured Design)：首先建立现实世界的模型，再确定系统的功能需求。以事件为驱动，基于进程的开发方法。

(3) 支持程序开发的形式化方法(基于模型的方法)——维也纳方法。

将软件系统当做模型来给予描述，把软件的输入、输出看作模型对象，把这些对象在计算机内的状态看做该模型在对象上的操作。

(4) 面向对象开发方法

- 基本出发点是尽可能按照人类认识世界的方法和思维方式来分析和解决问题。
- 面向对象开发方法包括面向对象分析、面向对象设计、面向对象实现。

2) 软件项目管理

软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

3) 软件工程中的技术复审和管理复审

每阶段结束前要进行技术复审和管理复审。

(1) 技术复审是从技术角度确保质量，降低软件成本(尽早发现问题)。审查过程包括准备(如成立审查小组)、简要介绍情况、阅读被审文档、开审查会、返工、复查等。

(2) 管理复审主要是从管理的角度对成本、进度、经费等进行复审，以保证项目正常开展。

在软件工程理论的指导下，发达国家已经建立起较为完备的软件工业化生产体系，形成了强大的软件生产能力。软件标准化与可重用性得到了工业界的高度重视，在避免重用劳动、缓解软件危机方面起到了重要作用。

1.3.2　软件生命周期

软件工程的传统解决途径强调使用生命周期方法学和各种结构分析及结构设计技术。

它们是在 20 世纪 70 年代为了应付应用软件日益增长的复杂程度、漫长的开发周期以及用户对软件产品经常不满意的状况而发展起来的。

人类解决复杂问题时普遍采用的一个策略就是各个击破，也就是对问题进行分解，然后再分别解决各个子问题的策略。软件工程实际上是从时间角度对软件开发和维护的复杂问题进行分解，把软件生存的漫长周期或把用户的要求转变成软件产品的过程。

1. 什么是软件生命周期

同任何事物一样，软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，一般称为软件生命周期(又称软件生存周期)。把整个软件生命周期划分为若干阶段，使得每个阶段有明确的任务，使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常，软件生命周期包括可行性分析与开发项计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护等活动，可以将这些活动以适当的方式分配到不同的阶段去完成。

这种按时间划分的思想方法是软件工程中的一种思想原则，即按部就班、逐步推进，每个阶段都要有定义、工作、审查、形成文档，以供交流或备查，以提高软件的质量。但随着新的面向对象设计方法和技术的成熟，软件生命周期设计方法的指导意义正在逐步减小。

2. 软件生命周期的阶段划分

1) 软件生命周期的阶段概念

可以从以下几个方面了解软件生命周期的阶段概念。

(1) 采用生命周期方法开发软件的时候，从对任务的抽象逻辑分析开始，一个阶段一个阶段地进行开发。

(2) 前一个阶段任务的完成是开始进行后一个阶段工作的前提和基础，而后一阶段任务的完成通常是使前一阶段提出的解法更进一步具体化，加入了更多的物理细节。

(3) 每一个阶段的开始和结束都有严格标准，对于任何两个相邻的阶段而言，前一阶段的结束标准就是后一阶段的开始标准。

(4) 在每一个阶段结束之前都必须进行正式、严格的技术审查和管理复审，从技术和管理两方面对这个阶段的开发成果进行检查，通过之后这个阶段才算结束(如果检查通不过，则必须进行必要的返工，并且返工后还要再经过审查)。

(5) 审查的一条主要标准就是每个阶段都应该交出最新式的(即和所开发的软件完全一致的)、高质量的文档资料，从而保证在软件开发工程结束时有完整、准确的软件配置交付使用。

(6) 文档是通信的工具，它们清楚、准确地说明了到这个时候为止，关于该项工程已经知道了什么，同时确立了下一步工作的基础。此外，文档也起备忘录的作用，如果文档不完整，那么一定是某些工作忘记做了，在进入生命周期的下一阶段之前，必须补足这些遗漏的细节。在完成生命周期每个阶段的任务时，应该采用适合该阶段任务特点的系统化的技术方法，如结构分析或结构设计技术。

2) 软件生命周期阶段划分的意义

软件生命周期阶段划分具有以下意义：

(1) 软件生命周期划分成若干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度。

(2) 在软件生命周期的每个阶段都采用科学的管理技术和良好的技术方法，而且在每个阶段结束之前都从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发工程的全过程以一种有条不紊的方式进行，保证了软件的质量，特别是提高了软件的可维护性。

总之，采用软件工程方法论可以大大提高软件开发的成功率，软件开发的生产率也能明显提高。

3) 软件生命周期阶段划分的方法

目前划分软件生命周期阶段的方法有许多种，软件规模、种类、开发方式、开发环境以及开发时使用的方法论都影响软件生命周期阶段的划分。在划分软件生命周期的阶段时应该遵循的一条基本原则就是使各阶段的任务彼此间尽可能相对独立，同一阶段各项任务的性质尽可能相同，从而降低每个阶段任务的复杂程度，简化不同阶段之间的联系，有利于软件开发工程的组织管理。一般说来，软件生命周期由软件定义、软件开发和软件维护三个时期组成，每个时期又进一步划分成若干个阶段。

(1) 软件定义时期的任务

软件定义时期的任务是：①确定软件开发工程必须完成的总目标；②确定工程的可行性，导出实现工程目标应该采用的策略及系统必须完成的功能；③估计完成该项工程需要的资源和成本，并且制定工程进度表。

这个时期的工作通常又称为系统分析，由系统分析员负责完成。软件定义时期通常进一步划分成三个阶段，即问题定义、可行性研究和需求分析。

(2) 开发时期的主要任务

开发时期的主要任务是具体设计和实现在前一个时期定义的软件，它通常由下述四个阶段组成：总体设计、详细设计、编码和单元测试、综合测试。

(3) 维护时期的主要任务是使软件持久地满足用户的新需求。具体地说：

① 当软件在使用过程中发现错误时应该加以改正；

② 当环境改变时应该修改软件以适应新的环境；

③ 当用户有新要求时应该改进软件以满足用户的新需求。通常对维护时期不再进一步划分阶段，但是每一次维护活动本质上都是一次压缩和简化的定义和开发过程。

软件生命周期中软件项目管理是自始至终的，软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

统计数据表明，大多数软件开发项目的失败，并不是软件开发技术方面的原因。它们的失败是由于不适当的管理造成的。遗憾的是，尽管人们对软件项目管理重要性的认识有所提高，但在软件管理方面的进步远比在设计方法学和实现方法学上的进步小，至今还提不出一套管理软件开发的通用指导原则。

3. 软件生命周期模型

任何软件都是从最模糊的概念开始的。从概念提出的那一刻开始，软件产品就进入了软件生命周期。在经历需求分析、系统设计、实现、部署后，软件将被使用并进入维护阶

段，直到最后由于缺少维护费用而逐渐消亡。这样的一个过程，称为生命周期模型(Life Cycle Model)。

典型的几种生命周期模型包括瀑布模型、迭代式模型、快速原型模型、增量模型、螺旋模型等。

1) 瀑布模型

瀑布模型由于酷似瀑布而闻名。在该模型中，首先确定需求，并接受客户和软件质量保证(Software Quality Assurance，SQA)小组的验证。然后拟定规格说明，同样通过验证后，进入计划阶段。可以看出，瀑布模型中至关重要的一点是：只有当一个阶段的文档已经编制好并获得软件质量保证小组的认可后，才可以进入下一个阶段。这样，瀑布模型通过强制性的要求提供规约文档来确保每个阶段都能很好地完成任务。但是实际上往往难以办到，因为整个模型几乎都是以文档驱动的，这对于非专业的用户来说是难以阅读和理解的。但对于应用结构化软件开发方法的大型软件系统的开发，瀑布模型有其天生的优势，如图 1-1 所示。

2) 迭代式模型

迭代式模型是 RUP(Rational Unified Process，统一软件开发过程，简称统一软件过程)推荐的周期模型。在 RUP 中，迭代就是为了完成一定的阶段性目标而从事的一系列开发活动，在每个迭代开始前都要根据项目当前的状态和所要达到的阶段性目标制定迭代计划，整个迭代过程包含需求分析、设计、实施(编码)、部署、测试等各种类型的开发活动，迭代完成之后需要对迭代完成的结果进行评估，并以此为依据来制定下一次迭代的目标，如图 1-2 所示。



图 1-1　瀑布模型示意图　　　图 1-2　迭代式模型示意图

3) 快速原型模型

快速原型(Rapid Prototype)模型在功能上等价于产品的一个简单原型。瀑布模型的缺点就在于不够直观，快速原型法就解决了这个问题。一般来说，根据客户的需要在很短的时间内满足用户的最迫切需要，完成一个可以演示的产品。这个产品只是实现部分的(最重要的)功能。它最重要的目的是确定用户的真正需求，如图 1-3 所示。

4) 增量模型

增量模型是一种非整体开发的模型，是瀑布模型的顺序特征和快速原型模型的迭代特征相结合的产物。该模型具有较大的灵活性，适合于软件需求不明确、设计方案有一定风

险的软件项目。

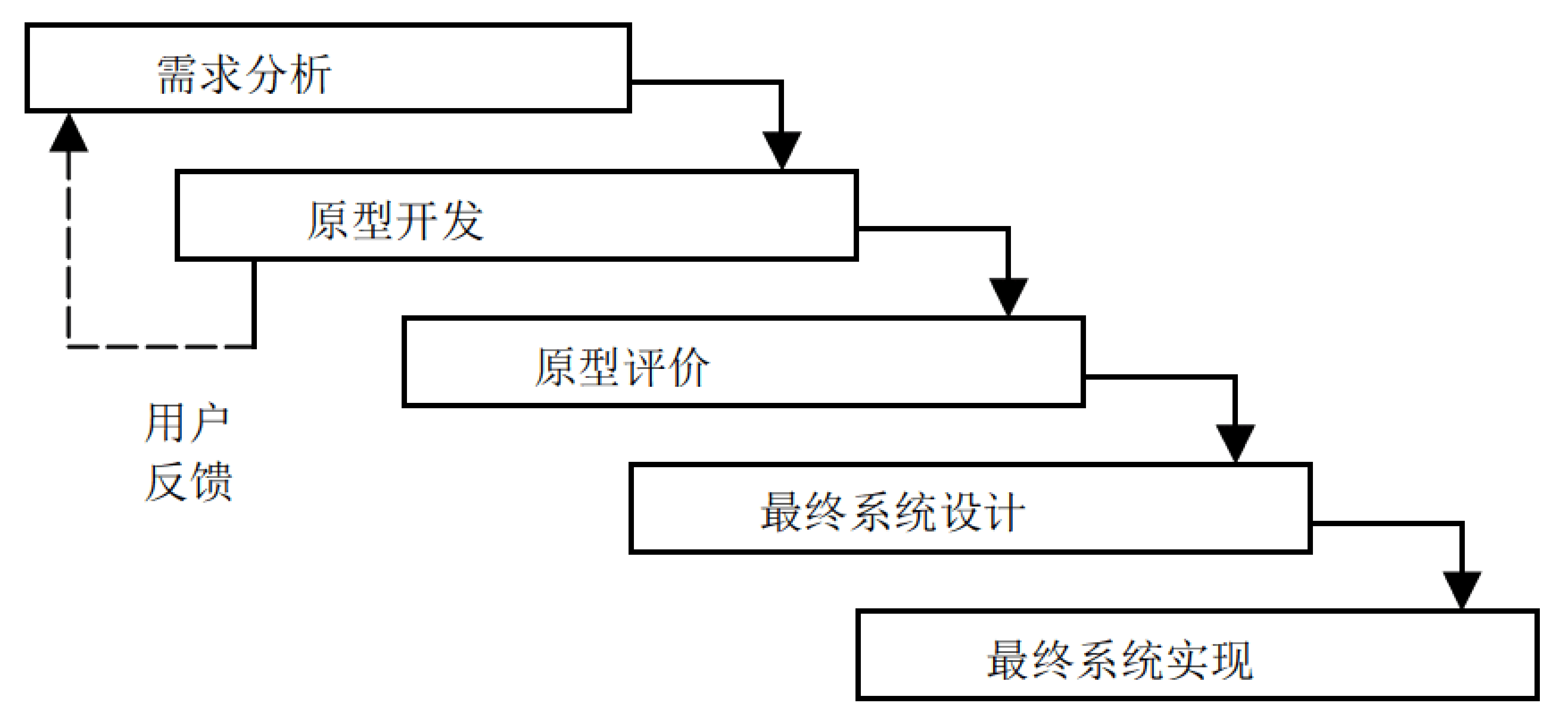


图 1-3　快速原型模型示意图

增量模型的特点是：在前面增量的基础上开发后面的增量，每个增量的开发可用瀑布或快速原型模型迭代的思路。其优点是：如果在项目既定的商业要求期限不可能找到足够的开发人员，这种情况下增量模型显得特别有用。早期的增量可以由少量的人员实现。同时，增量模型可以规避技术风险，如图 1-4 所示。

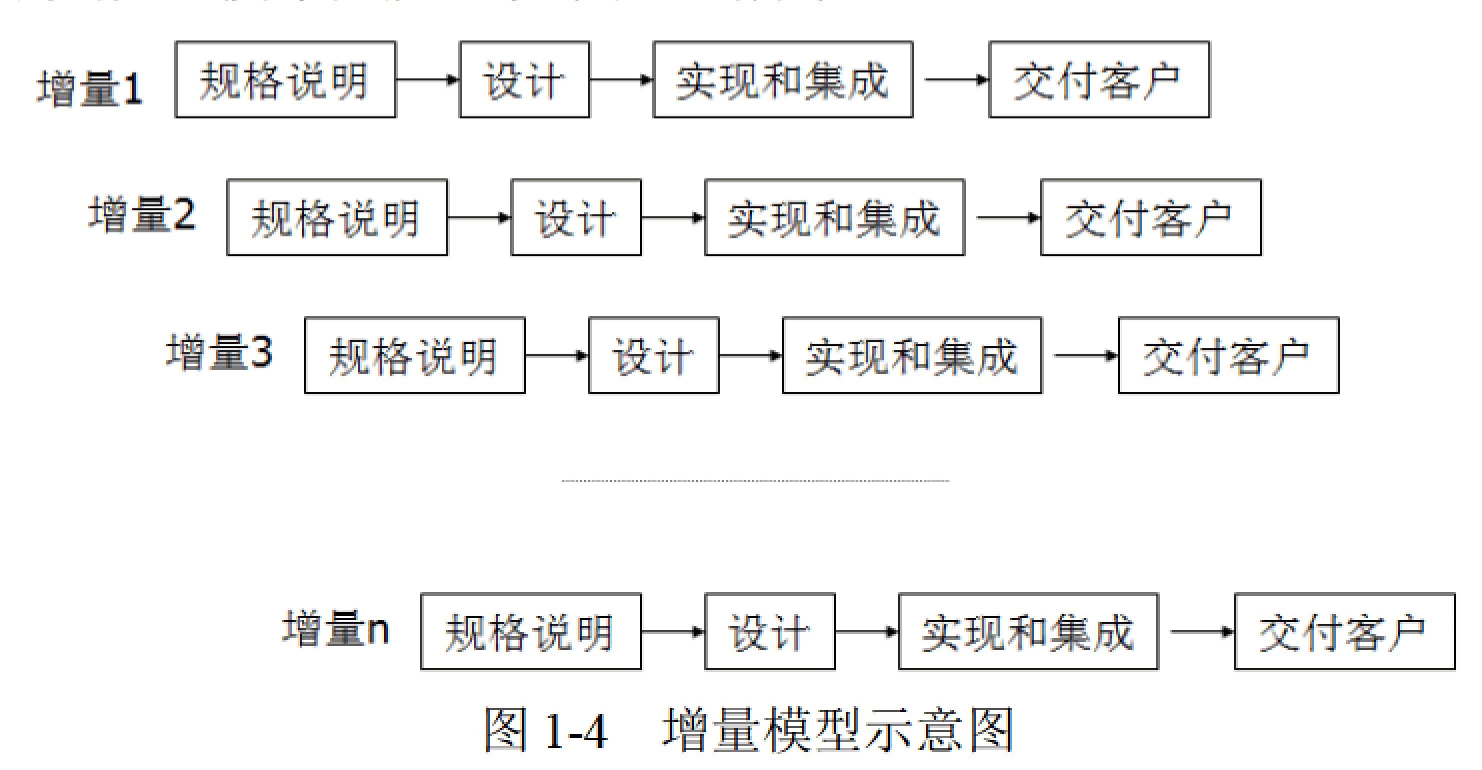


图 1-4　增量模型示意图

5) 螺旋模型

螺旋模型的基本思想是，使用原型及其他方法以尽可能地降低风险。理解这种模型的一种简易方法是，把它看作在每个阶段之前都增加了风险分析过程的快速原型模型，如图 1-5 所示。

螺旋模型将瀑布模型与快速原型模型结合起来，加入了两种模型均忽略的风险分析，弥补了这两种模型的不足。螺旋模型是一种风险驱动的模型。它将开发过程分为几个螺旋周期，每个螺旋周期大致和瀑布模型相符合。螺旋模型适合于大型软件的开发。

螺旋模型是一种迭代式模型，每迭代一次，螺旋线就前进一周。当项目按照顺时针方向沿螺旋移动时，每一个螺旋周期包含了风险分析，并且按以下 4 个步骤来进行：

(1) 确定目标，选定方案，设定约束条件，选定完成本周期所定目标的策略。

(2) 分析该策略可能存在的风险。必要时通过建立一个原型来确定风险的大小，然后据此决定是按原定目标执行，还是修改目标或终止项目。

(3) 在排除风险之后，实现本螺旋周期的目标。例如，第一圈可能产生产品的规格说明，第二圈可能产生产品设计等。

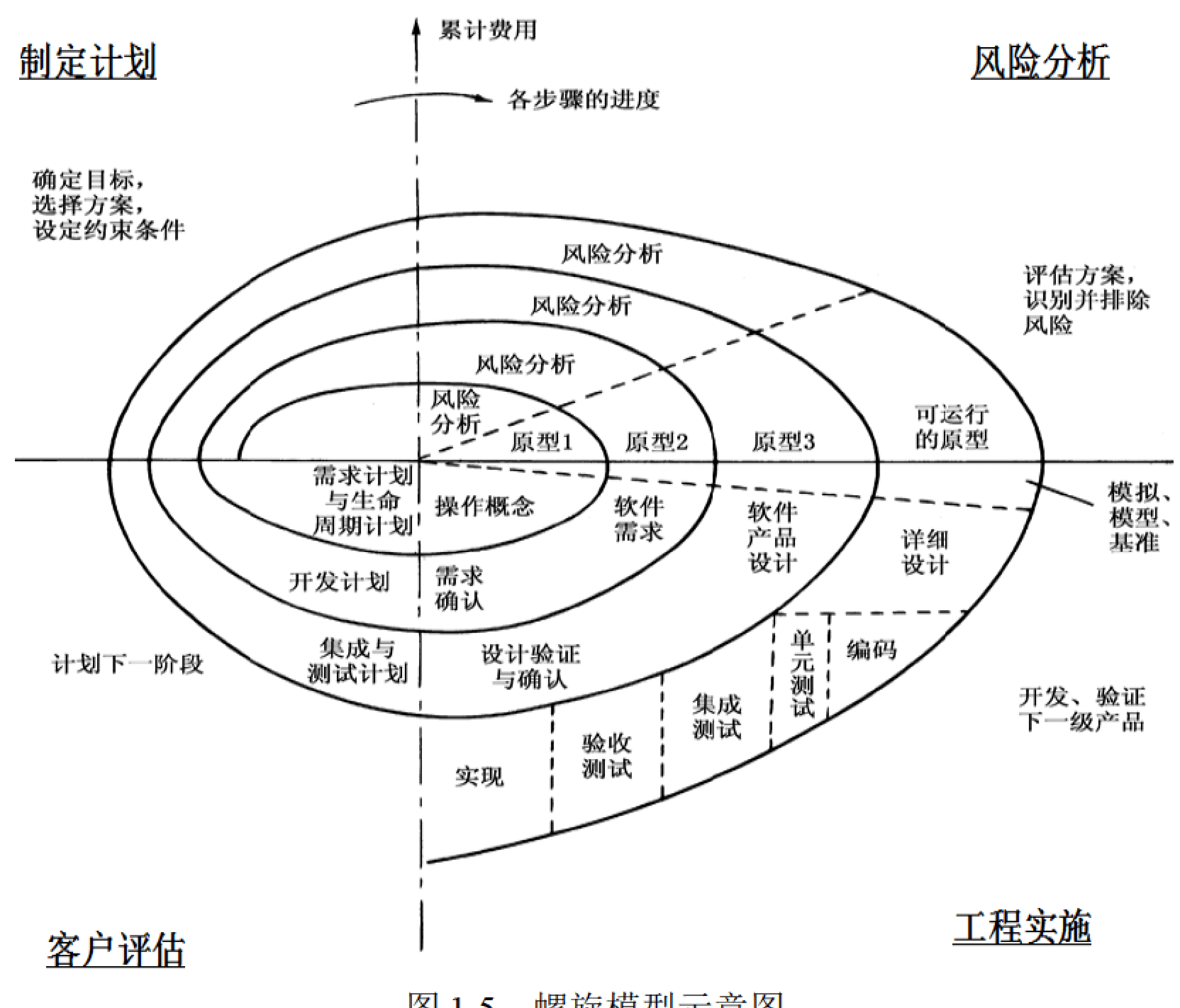


图 1-5　螺旋模型示意图

(4) 最后一步是评价前一步的结果，并且计划下一轮的工作。

螺旋模型的优点是：结合瀑布模型和原型模型的优点，风险分析可使一些极端困难的问题和可能导致费用过高的问题被更改或取消。

螺旋模型的缺点是：螺旋模型开发的成败，很大程度上依赖于风险评估的成败。需要开发人员具有相当丰富的风险评估经验和专业知识。

螺旋模型的一般使用场合：需求不能完全确定，同时又存在技术、资金或开发时间等风险因素的大型开发项目。

软件生命周期模型的发展实际上体现了软件工程理论的发展。在早期，软件的生命周期处于无序、混乱的情况。一些人为了能够控制软件的开发过程，就把软件开发严格地区分为多个不同的阶段，并在阶段间加上严格的控制和审查，确保质量。否则，就像图 1-6 所示的那样，在软件生命周期中发现问题和解决问题而付出的代价，将会随着时间和阶段的变化成倍或呈数量级增加。

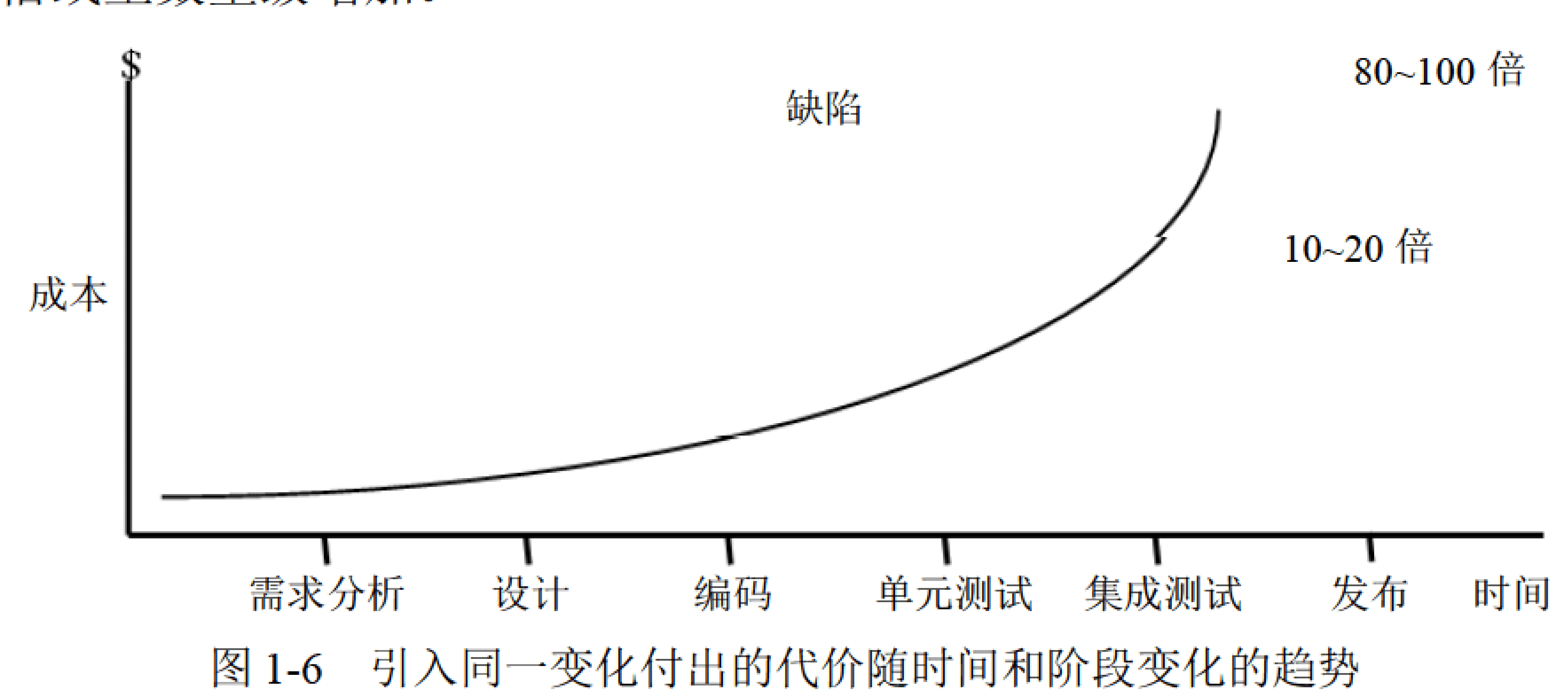


图 1-6　引入同一变化付出的代价随时间和阶段变化的趋势

1.3.3　敏捷开发过程

自从 20 世纪 70 年代软件工程产生以来，人们提出了很多软件开发方法，这些方法大都强调软件开发过程中必须遵守某些严格的规定。而且随着技术的发展，提出了在对需求和技术不断变化的情况下实现快速软件开发的要求。在 20 世纪 90 年代后期，一些软件开发人员开始强调灵活性在软件开发过程中所发挥的作用，提出一系列新的软件开发方法，这就是敏捷方法或敏捷开发。

1. 敏捷开发

敏捷开发被认为是软件工程的一次重要的发展。它强调软件开发应当能够对未来可能出现的变化和不确定性做出全面反应。

敏捷开发的总体目标是通过尽可能早地、持续地对有价值软件的交付，使客户满意。很多客户都有一些随着时间变化的业务需求，不仅表现在新发现的需求上，也表现在对市场变化做出反应的需求上。通过在软件开发过程中加入灵活性，敏捷开发可以使用户能够在开发周期的后期增加或改变需求。

敏捷开发主要是用于需求模糊或快速变化的前提下、小型开发团队的软件开发活动。敏捷开发能够在保证软件开发成功的前提下，尽量减少开发过程中的活动和产品，做到刚刚好，从而在满足所需的软件质量要求的前提下，力求提高开发的效率。

敏捷开发强调：

- 注重个人及互动胜于过程和工具
- 注重可用的软件胜于详尽的文档
- 注重客户协作胜于合同谈判
- 注重响应变化胜于恪守计划

敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。

敏捷开发是针对传统的瀑布开发模型的弊端而产生的一种新的开发模式，是一种面临迅速变化的需求快速开发软件的能力，目标是提高开发效率和响应能力。为达到该目标，敏捷开发定义了 12 条原则：

(1) 最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。

(2) 即使在开发的后期，也不拒绝需求变更(敏捷过程利用变更为客户创造竞争优势)。

(3) 经常交付可工作的软件(交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好)。

(4) 在整个项目开发期间，业务人员和开发人员最好天天在一起工作。

(5) 强化激励机制，为受激励的个人构建项目(为他们提供所需的环境和支持，并且信任他们能够完成工作)。

(6) 在团队内部，最富有效果和效率的信息传递方法是面对面交谈。

(7) 可工作的软件是进度的首要度量标准。

(8) 敏捷过程提倡可持续的开发速度(责任人、开发者和用户应该保持一种长期、稳定

的开发速度)。

(9) 不断地关注优秀的技能和好的设计，增强敏捷能力。

(10) 尽量简化要做的工作。

(11) 好的架构、需求和设计出自于组织团队自身。

(12) 每隔一定时间，团队要反省如何更有效地工作，并相应地调整自己的行为。

敏捷方法有很多具体的方法，常用的敏捷方法有七种：

1) XP

XP(eXtreme Programming，极限编程)的思想源自 Kent Beck 和 Ward Cunningham 在软件项目中的合作经历。XP 注重的核心是沟通、简明、反馈和勇气。因为知道计划永远赶不上变化，XP 不需要开发人员在软件开始初期做出很多的文档。XP 提倡测试先行，以将后面出现 bug 的概率降至最低。

2) SCRUM

SCRUM 是一种迭代式的增量化过程，用于产品开发或工作管理。它是一种可以集合各种开发实践的经验化过程框架。SCRUM 中发布产品的重要性高于一切。

该方法由 Ken Schwaber 和 Jeff Sutherland 提出，旨在寻求充分发挥面向对象和构件技术的开发方法，是对迭代式面向对象方法的改进。

3) Crystal Methods

Crystal Methods(水晶方法族)由 Alistair Cockburn 在 20 世纪 90 年代末提出。之所以是一个方法系列，是因为他相信不同类型的项目需要不同的方法。虽然水晶方法族不如 XP 那样的高产出效率，但会有更多的人能够接受并遵循它。

4) FDD

FDD(Feature-Driven Development，特性驱动开发)由 Peter Coad、Jeff de Luca、Eric Lefebvre 共同开发，是一套针对中小型软件开发项目的开发模式。此外，FDD 是一个模型驱动的快速迭代开发过程，它强调的是简化、实用、易于被开发团队接受，适用于需求经常变动的项目。

5) ASD

ASD(Adaptive Software Development，自适应软件开发)由 Jim Highsmith 在 1999 年正式提出。ASD 强调开发方法的适应性(Adaptive)，这一思想来源于复杂系统的混沌理论。ASD 不像其他方法那样有很多具体的实践做法，它更侧重为 ASD 的重要性提供最根本的基础，并从更高的组织和管理层次来阐述开发方法为什么要具备适应性。

6) DSDM

DSDM(Dynamic System Development Method，动态系统开发方法)是众多敏捷开发方法中的一种，它倡导以业务为核心，快速而有效地进行系统开发。实践证明，DSDM 是成功的敏捷开发方法之一。在英国，由于其在各种规模的软件组织中的成功，已成为应用最为广泛的快速应用开发方法。

DSDM 不但遵循敏捷方法的原理，而且非常适合那些前期有较好传统开发方法基础的软件组织。

7) 轻量型 RUP

RUP 其实是一个过程的框架，它可以包容许多不同类型的过程，Craig Larman 极力主张以敏捷方式使用 RUP。他的观点是：目前如此多的努力以推进敏捷方法，只不过是在接受能被视为 RUP 的主流面向对象开发方法而已。

2. 敏捷开发过程

敏捷可用于任何软件过程，实现要点是将软件过程设计为如下方式：允许项目团队调整并合理安排任务，理解敏捷开发方法的易变性并制定计划，精简并维持最基本的工作产品，强调增量交付策略，快速向客户提供适应产品类型和运行环境的可运行软件。因此，敏捷过程很容易适应变化并迅速做出自我调整，在保证质量的前提下，做到文档、度量适度。

下面我们以极限编程(eXtreme Programming，XP)为例，介绍敏捷开发过程。

XP 使用面向对象方法作为推荐的开发范型。XP 包含了策划、设计、编码和测试 4 个框架活动的规则和实践。图 1-7 描述了 XP 过程，并指出了与各框架活动相关的关键概念和任务。

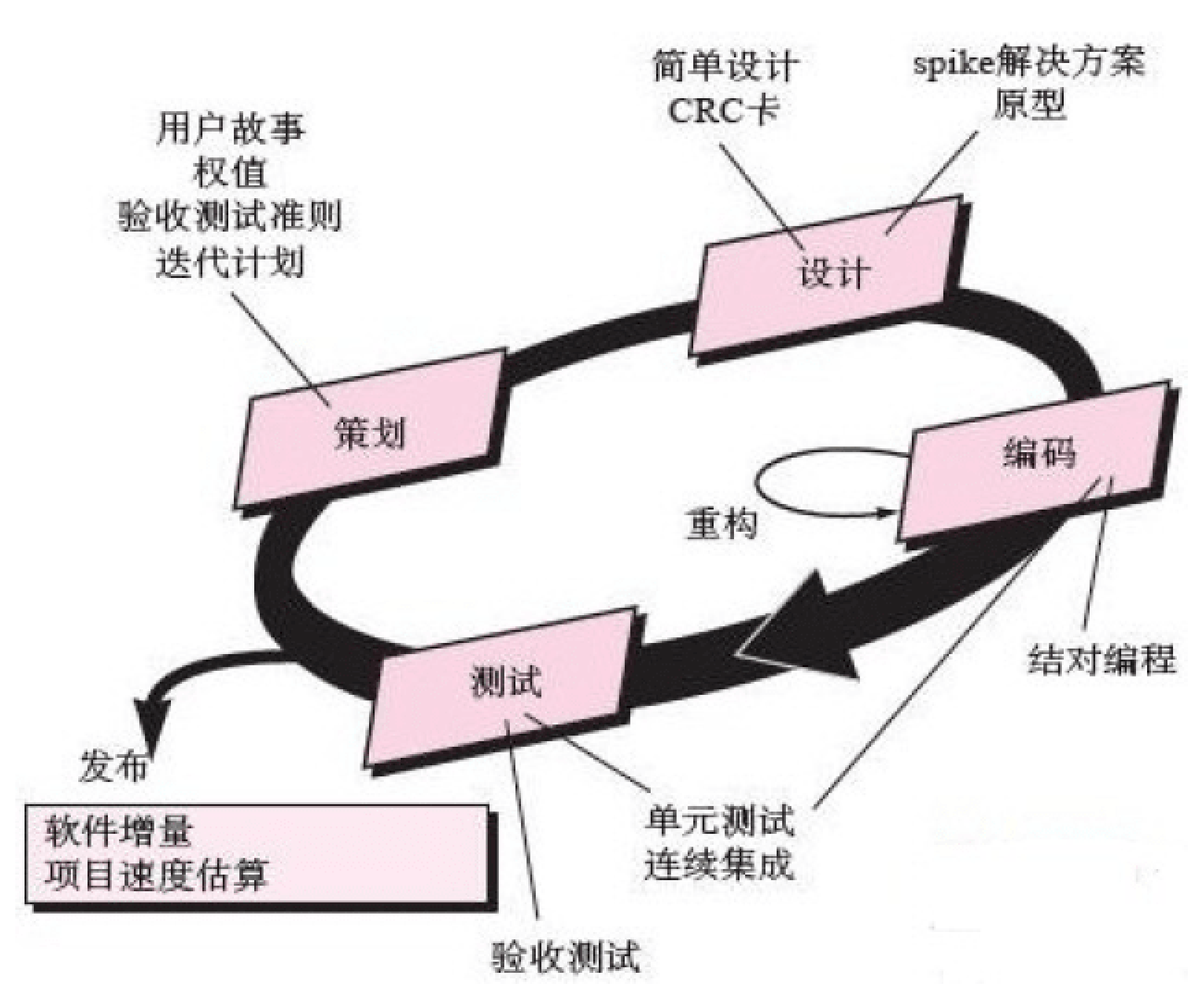


图 1-7　极限编程过程

1) 策划

策划活动开始于建立一系列描述待开发软件的必要特征与功能要求。每个功能要求由客户书写并置于一张索引卡上，客户根据对应特征或功能的全局业务价值标明权值(即优先级)。XP 团队成员评估每一个功能并给出以开发周数为度量单位的成本。如果某个功能的开发成本超过 3 周，将请客户对该功能进一步细分，重新赋予权值并计算成本。新的功能要求可以在任何时刻书写。

客户和 XP 团队共同决定如何把功能分组，并置于 XP 团队将要开发的下一个发行版本中。一旦形成关于一个发行版本的基本承诺，XP 团队将按以下三种方式之一对待开发的功能进行排序：

① 所有选定功能将在几周之内尽快实现；

② 具有最高价值的功能将移到进度表的前面并首先实现；

③ 高风险功能将首先实现。

项目的第一个发行版本发布之后，XP 团队计算项目的速度。简言之，项目速度是第一个发行版本中实现的用户功能个数。项目速度将用于帮助建立后续发行版本的发布日期和进度安排；确定是否对整个开发项目中的所有功能有过分承诺。一旦发生过分承诺，就调整软件发行版本的内容或者改变最终交付日期。

在开发过程中，客户可以增加功能、改变功能要求的权值、分解或去掉功能。接下来由 XP 团队重新考虑所有剩余的发行版本，并相应修改计划。

2) 设计

XP 设计严格遵循保持简洁(Keep It Simple，KIS)原则，使用简单而不是复杂的表述。另外，设计为功能提供不多也不少的实现原则，不鼓励额外功能性设计。

XP 鼓励使用类、责任、协作者(Class-Responsibility-Card，CRC)卡作为有效机制，在面向对象语境中考虑软件、CRC 卡的确定，组织和当前软件增量相关的对象和类。CRC 卡也是作为 XP 过程一部分的唯一的设计工作产品。

如果在某个功能设计中碰到困难，XP 推荐立即建立这部分设计的可执行原型，实现并评估设计原型，目的是在真正的实现开始时降低风险，对可能存在设计问题的功能确认最初的估计。

3) 编码

在功能开发和基本设计完成之后，团队不应直接开始编码，而是开发一系列用于检测本次(软件增量)发布的包括所有功能的单元测试。一旦建立起单元测试，开发者就可以更集中精力于必须实现的内容以通过单元测试。不需要添加任何额外的东西(保持简洁)。一旦编码完成，就可以立即完成单元测试，可由此向开发者提供即时反馈。

XP 编码活动中的关键概念之一是结对编程。XP 推荐两个人面对同一台计算机共同为一个功能开发代码。这一方案提供实时解决问题和实时质量保证的机制，同时也使开发者能集中精力于手头的问题。实施中不同成员担任的角色略有不同。例如，一名成员考虑特定设计的详细编码实现，而另一名成员确保编码遵循特定的标准，生成的代码符合该功能要求的接口设计。

结对的两人完成所开发代码和其他工作集成。在有些情况下，这种集成工作由集成团队按日实施。在另外一些情况下，结对者自己负责集成，这种“连续集成”策略有助于避免兼容性和接口问题，建立能及早发现错误的“冒烟测试”环境。

4) 测试

在编码开始之前建立单元测试是 XP 方法的关键因素。所建立的单元测试应当使用一个可以自动实施的框架，这种方式支持代码修改之后的即时回归测试策略。

一旦将个人的单元测试组织到一个“通用测试集”，每天就可以进行系统的集成和确认测试。这可以为 XP 团队提供连续的进展指示，也可在一旦发生问题的时候及早提出预警。

XP 验收测试也称为客户测试，由客户确定，将着眼于客户可见的、可评审的系统级的特征和功能，验收测试根据本次软件发布中所实现的用户功能而确定。

习题和思考题

1. 简述软件的定义，软件具有什么样的特性？
2. 什么是软件危机？产生软件危机的原因是什么？如何消除？
3. 什么是软件工程？什么是软件生命周期？它们都包含哪些内容？
4. 软件工程涉及哪些概念和名词？它们的关系如何？如何解释？
5. 运用软件工程的理论、技术和方法能够为我们解决什么问题？
6. 如何理解软件开发工具和软件工程环境在软件工程中的作用？
7. 软件项目管理涉及哪些方面，它的必要性是什么？
8. 软件复审的目的是什么，我们怎样进行软件技术审查？软件技术审查和管理复审的作用是什么？
9. 什么是软件开发模型？软件开发模型有几种？各有什么特点？
10. 什么是敏捷开发？敏捷开发有哪些方法？其基本过程是怎样的？

第2章　软件测试基础

信息技术的飞速发展，使软件产品应用到社会的各个领域，软件产品的质量自然成为人们共同关注的焦点。不论软件的生产者还是使用者，均生存在竞争激烈的环境中，软件开发商为了占有市场，必须把产品质量作为企业的重要目标之一，以免在激烈的竞争中被淘汰出局。用户为了保证自己业务的顺利完成，当然希望选用优质的软件。质量不佳的软件产品不仅会使开发商的维护费用和用户的使用成本大幅增加，还可能产生其他的责任风险，造成公司信誉下降，继而冲击股票市场。在一些关键应用(如民航订票系统、银行结算系统、证券交易系统、自动飞行控制软件、军事防御和核电站安全控制系统等)中使用质量有问题的软件，还可能造成灾难性的后果。

事实上，对于软件来讲，还没有像银弹那样的东西。在软件开发过程中，不论采用什么技术和方法，软件开发者难免会因智力、逻辑的劳动方式常在工作中犯错误，从而使软件产品隐藏着许多错误和缺陷，规模大、复杂性高的软件更是如此。采用新的语言、先进的开发方式、完善的开发过程，可以减少错误的引入，但是不可能完全杜绝软件中的错误。这些错误有些是致命性的，如不排除，就会导致生命与财产的重大损失。这些引入的错误需要测试来找出，软件中的错误密度也需要测试来进行估计。

2.1　软件测试基本概念

测试是所有工程学科的基本组成单元。对于软件工程而言，软件测试是软件开发的重要组成部分，是软件工程的重要分支。软件测试是确保软件质量的重要一环，测试是手段，质量是目的，属于软件工程领域。自有程序设计的那天起测试就一直伴随着。统计表明，在典型的软件开发项目中，软件测试工作量往往占软件开发总工作量的40%以上。而在软件开发的总成本中，用在测试上的开销要占30%到50%。如果把维护阶段也考虑在内，讨论整个软件生命周期时，测试的成本比例也许会有所降低，但实际上维护工作相当于二次开发，乃至多次开发，其中必定还包含许多测试工作。因此，测试对于软件生产来说是必需的，问题是我们应该思考采用什么方法？如何安排测试？

2.1.1　软件测试发展史

软件测试是伴随着软件的产生而产生的。

1. 测试等同于调试

早期的软件开发过程中，那时软件规模都很小、复杂程度低，软件开发的过程混乱无序、相当随意，测试的含义比较狭窄，开发人员将测试等同于调试，目的是纠正软件中已

经知道的故障，常常由开发人员自己完成这部分工作。对测试的投入极少，测试介入也晚，常常是等到形成代码，产品已经基本完成时才进行测试。

直到 1957 年，软件测试才开始与调试区别开来，作为一种发现软件缺陷的活动。

2. 测试是一种发现软件缺陷的活动

由于一直存在着为了让我们看到产品在工作，就得将测试工作往后推一点的思想，潜意识里对测试的目的就理解为使自己确信产品能工作。测试活动始终滞后于开发活动，测试通常被作为软件生命周期中的最后一项活动而进行。当时也缺乏有效的测试方法，主要依靠错误推测(Error Guessing)来寻找软件中的缺陷。因此，大量软件交付后，仍存在很多问题，软件产品的质量无法保证。

因此，我们可以说，在软件工程建立之前的 20 世纪 60 年代，软件测试是为表明程序正确而进行的测试。

到了 20 世纪 70 年代，这个阶段开发的软件仍然不复杂，但人们已开始思考软件开发流程的问题，尽管对软件测试的真正含义还缺乏共识，但这一词条已经频繁出现，一些软件测试的探索者建议在软件生命周期的开始阶段就根据需求制定测试计划，这时也涌现出一批软件测试的宗师，Bill Hetzel 博士就是其中的领导者。

1972 年，软件测试领域的先驱 Bill Hetzel 博士(代表著作 *The Complete Guide to Software Testing*)，在美国的北卡罗来纳(North Carolina)大学组织了历史上第一次正式的关于软件测试的会议。

1973 年，Bill Hetzel 博士给软件测试下了这样的定义：就是建立一种信心，认为程序能够按预期的设想运行。

1975 年，John Good Enough 和 Susan Gerhart 在 IEEE 上发表了一篇名为“测试数据选择的原理”的文章，首次提出了软件测试理论。同年 Huang 全面讨论了测试过程和测试准则，从此软件测试被确定为一种研究方向。

1979 年，Glenford Myers 在《软件测试的艺术》一书中提出测试的目的是证伪，即测试是为发现错误而执行的一个程序或系统的过程。

3. 现代软件测试定义的产生

20 世纪 80 年代早期，质量的号角开始吹响，各个软件企业开始建立 QA、SQA 部门及其演化流程，软件测试的定义发生了改变。测试不单纯是一个发现错误的过程，而且包含软件质量评价的内容，人们制定了各类标准。

1981 年，Bill Hetzel 博士开设了一门公共课——结构化软件测试(Structured Software Testing)。1983 年，他在出版的《软件测试完全指南》一书中对 1973 年他所给出的软件测试定义进行了修订：测试是以评价程序或系统的属性及其功能的各种活动。

1988 年，David Gelperin 和 Bill Hetzel 在 *Communications of the ACM* 上发表了文章 *The Growth of Software Testing*，介绍系统化的测试和评估流程。

20 世纪 80 年代后期，Paul Rook 提出了著名的软件测试的 V 模型，旨在改进软件开发的效率和效果。从此，软件测试模型与软件测试标准的研究也随着软件工程的发展而越来越深入。

20 世纪 90 年代，测试工具盛行起来。

1996 年提出了测试能力成熟度模型(TCMM，Testing Capability Maturity Model)、测试支持度模型(TSM，Testability Support Model)、测试成熟度模型(TMM，Testing Maturity Model)。

到了 2002 年，Rick 和 Stefan 在《系统的软件测试》一书中对软件测试做了进一步定义：测试是为了度量和提高被测软件的质量，对测试软件进行工程设计、实施和维护的整个生命周期过程。

2.1.2　软件测试的定义

1. Bill Hetzel 博士的软件测试定义

软件测试的定义最早是 Bill Hetzel 博士在 1973 年提出来的。他认为软件测试的目的是建立一种信心，认为程序能够按预期的设想运行。后来在 1983 年他又将定义修订为：评价程序和系统的属性或功能，并确定它是否达到预期的结果。软件测试就是以此为目的的各种行为。在他的定义中，“预期的结果”其实就是我们现在所说的用户需求或功能设计。他还把软件的质量定义为符合要求。他的思想的核心观点是：测试方法是试图验证软件是工作的，即软件的功能是按照预先的设计执行的，以正向思维，针对软件系统的所有功能点，逐个验证正确性。软件测试业界把这种方法看作软件测试的第一类方法。

2. Glenford Myers 的软件测试定义

尽管如此，Bill Hetzel 的这一方法还是受到很多业界权威的质疑和挑战。代表人物是前面提到的 Glenford Myers。他认为测试不应该着眼于验证软件是工作的，相反应该首先认定软件是有错误的，然后用逆向思维去发现尽可能多的错误。他还从心理学的角度论证，如果将验证软件是工作的作为测试的目的，非常不利于测试人员发现软件的错误。于是，他在我们前面提到的《软件测试的艺术》一书中提出了他对软件测试的定义：测试是为发现错误而执行的一个程序或系统的过程。这个定义也被业界所认可，经常被引用。除此之外，Myers 还给出了与测试相关的三个重要观点，那就是：①测试是为了证明程序有错，而不是证明程序无错误；②一个好的测试用例在于它能发现至今尚未发现的错误；③一次成功的测试是发现了至今尚未发现的错误的测试。这就是软件测试的第二类方法，简单地说就是验证软件是不工作的，或者说是有错误的。Myers 认为，一次成功的测试必须是发现 bug 的测试，不然就没有价值。这就如同一个病人(假定此人确实有病)，到医院做一项医疗检查，结果各项指标都正常，那说明该项医疗检查对于诊断该病人的病情是没有价值的，是失败的。

Myers 提出的测试的目的是证伪这一概念，推翻了过去为表明软件正确而进行测试的错误认识，为软件测试的发展指出了方向，软件测试的理论、方法在之后得到了长足发展。第二类软件测试方法在业界也很流行，受到很多学术界专家的支持。

然而，对测试的目的是证伪这一概念的理解也不能太过于片面。在很多软件工程学、软件测试方面的书籍中都提到一个概念：测试的目的是寻找错误，并且是尽最大可能找出最多的错误。这很容易让人们认为测试人员就是挑毛病的，而由此带来诸多问题，包括大家熟悉的 Ron Patton 在《软件测试》一书中就有如下明确而简洁的定义：软件测试人员的

目标是找到软件缺陷，尽可能早一些，并确保其得以修复。这样的定义具有一定的片面性，带来的结果是：①若测试人员以发现缺陷为唯一目标，而很少去关注系统对需求的实现，测试活动往往会存在一定的随意性和盲目性；②若有些软件企业接受了这样的方法，以 bug 数量作为考核测试人员业绩的唯一指标，也不太科学。

总的来说，第一类测试可以简单抽象地描述为这样的过程：在设计规定的环境下运行软件的功能，将其结果与用户需求或设计结果相比较，如果相符，则测试通过，如果不相符，则视为 bug。这一过程的终极目标是将软件的所有功能在所有设计规定的环境下全部运行并通过。在软件行业中一般把第一类方法奉为主流和行业标准。第一类测试方法以需求和设计为本，因此有利于界定测试工作的范畴，更便于部署测试的侧重点，加强针对性。这一点对于大型软件的测试，尤其是在有限的时间和人力资源情况下显得格外重要。

而第二类测试方法与需求和设计没有必然的关联，更强调测试人员发挥主观能动性，用逆向思维方式，不断思考开发人员理解的误区、不良的习惯、程序代码的边界、无效数据的输入以及系统各种的弱点，试图破坏系统、摧毁系统，目标就是发现系统中各种各样的问题。这种方法往往能够发现系统中存在的更多缺陷。

3. 现代软件测试定义

到了 20 世纪 80 年代初期，软件和 IT 行业开始了大发展，软件趋向大型化、高复杂度，软件的质量越来越重要。这个时候，一些软件测试的基础理论和实用技术开始形成，并且人们开始为软件开发设计了各种流程和管理方法，软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程，以结构化分析与设计、结构化评审、结构化程序设计以及结构化测试为特征。人们还将质量的概念融入其中，软件测试的定义发生了改变，测试不单纯是一个发现错误的过程，而且将测试作为软件质量保证(SQA)的主要职能，包含软件质量评价的内容。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。软件测试已有了行业标准(IEEE/ANSI)，1983 年 IEEE 提出的软件工程术语中给软件测试下的定义是：使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。这个定义明确指出：软件测试的目的是检验软件系统是否满足需求。它再也不是一次性的，而且只是开发后期的一项活动，而是与整个开发流程融合成一体。软件测试已成为一个专业，需要运用专门的方法和手段，需要专门人才和专家来承担。

事实上，人们目前对软件测试的认识从广义上讲，软件测试是指软件产品生命周期内所有的检查、评审和确认活动，如设计评审、系统测试；从狭义上讲，软件测试是对软件产品质量的检验和评价。它一方面检查软件产品中存在的质量问题，同时对产品质量进行客观的评价。基于这些认识，我们可以给出软件测试的定义：软件测试就是在软件投入运行前，对软件需求分析、设计规格说明和编码的最终复查，是软件质量保证的关键步骤。在该定义下，借鉴 Bill Hetzel 和 Glenford Myers 有关软件测试的思想，我们可以引出如下重要概念以满足测试工作的多种需要：

(1) 软件测试是对程序或系统能否完成特定任务建立信心的过程，也是帮助识别开发完成(中间或最终的版本)的计算机软件(整体或部分)的正确度(correctness)、完全度(completeness)和质量(quality)的软件过程；

(2) 软件测试就是为了发现程序中的错误而分析或执行程序的过程，或者说是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例，并利用这些测试用例运行程序，以发现程序错误的过程；

(3) 软件测试的目标在于尽可能地发现错误(缺陷)；

(4) 软件测试的目的在于鉴定程序或系统的属性或功能的各种活动，是软件质量的一种度量，是 SQA(Software Quality Assurance)的重要子域；

(5) 使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求(遗漏、超出)或是弄清预期结果与实际结果之间是否有差别。

软件测试在软件生命周期中横跨两个阶段：

① 单元测试阶段：编写出每个模块之后，就对它做必要的测试。

② 综合测试阶段：结束单元测试后进行的测试，如系统测试、验收测试等。

2.1.3　软件测试的目的

用户普遍希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品；软件开发者则希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心；而软件测试者则是替用户受过。

早期人们做测试，所期望达到的目的有几点：①测试是程序的执行过程，目的在于发现错误；②一个好的测试用例在于能发现至今尚未发现的错误；③一次成功的测试是发现了至今尚未发现的错误的测试。

1. 当前关于软件测试目的的几种观点

1) 软件测试的目的是尽可能发现并改正被测试软件中的错误，提高软件的可靠性

这个观点听起来很正确，但用它指导测试会带来很多问题。比如有的组织用发现的 bug 数来衡量测试人员的业绩，其实这就是这种测试目的论在后面作祟，其结果：①有一些不够敬业的测试人员会找来一些无关痛痒的 bug 来充数，结果许多时间会被浪费在这些无关痛痒的 bug 上(其实应该修复，何时修复，严重程度如何，优先级是什么，等等)；②测试人员会花很大力气设计一些复杂的测试用例去发现一些迄今尚未发现的缺陷，而不关心这些缺陷在实际用户的使用过程中是否会发生，从而浪费了大量的宝贵时间。究其根源，就是因为对测试目的的这种错误理解造成的，为什么这么说呢？因为软件里 bug 的数量是无从估计的，那么如果测试的目的是找 bug，那么测试工作将变成一项无法完成，也无法衡量进度且部分无效的工作(因为有些 bug 在实际的运行过程中根本不会发生)。

2) 软件测试的目的就是保证软件质量

这个观点也看似正确，但实际上混淆了测试和质量保证工作的边界。软件质量要素有很多，包括可理解性、简洁性、可移植性、一致性、可维护性、可测试性、可用性、有效性、安全等，所以，软件质量保证和测试其实关注的方向是不同的。

实际上，正确的软件测试目的的概念在 IEEE 1983 年提出的软件测试定义中已明确给定：使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是否弄清预期结果与实际结果之间的差别。所以，软件测试的目的应该是验证需求，

bug(预期结果与实际结果之间的差别)是这个过程中的产品而非目标。测试人员在软件产品投入使用之前对需求进行测试检验或验证，把尽可能多的问题在产品交给用户之前发现并改正。

2. 软件测试一般要达到的具体目标

1) 确保产品完成了它所承诺或公布的功能，并且所有用户可以访问到的功能都有明确的书面说明

产品缺少明确的书面文档，是厂商一种短期行为的表现，也是一种不负责任的表现。所谓短期行为，是指缺少明确的书面文档既不利于产品最后的顺利交付，容易与用户发生矛盾，影响厂商的声誉和将来与用户的合作关系；同时也不利于产品的后期维护，也使厂商支出超额的用户培训和技术支持费用。从长期利益看，这是很不划算的。

当然，书面文档的编写和维护工作对于使用快速原型法(RAD)开发的项目最为重要、最为困难，也是最容易被忽略的。

最后，书面文档的不健全甚至不正确，也是测试工作中遇到的最大和最头痛的问题，它的直接后果是测试效率低下、测试目标不明确、测试范围不充分，从而导致最终测试的作用不能充分发挥、测试效果不理想。

2) 确保产品满足性能和效率的要求

使用起来系统运行效率低(性能低)、用户界面不友好或用户操作不方便(效率低)的产品不能说是一个有竞争力的产品。实际上，用户最关心的不是你的技术有多先进、功能有多强大，而是他能从这些技术、这些功能中得到多少好处。也就是说，用户关心的是他能从中取出多少，而不是你已经放进去多少。

3) 确保产品是健壮的且适应用户环境

健壮性即稳定性，是产品质量的基本要求，尤其是在事务关键或时间关键的工作环境中。另外就是不能假设用户的环境(某些项目可能除外)。

总之，测试的目的是系统地找出软件中潜在的各种错误和缺陷，并能够证明软件的功能和性能与需求说明相符合。需要注意的是：测试不能表明软件中不存在错误，它只能说明软件中存在错误。

2.1.4　软件测试的原则

软件测试经过几十年的发展，测试界提出了很多软件测试的基本原则，为测试管理人员和测试人员提供了测试指南。软件测试原则非常重要，测试人员应该在测试原则的指导下进行测试活动。另外，软件测试的基本原则有助于测试人员进行高质量的测试，尽早、尽可能多地发现缺陷，并负责跟踪和分析软件中的问题，对存在的问题和不足提出质疑和改进，从而持续改进测试过程。

从朴素观点来说，对于相对复杂的产品或系统而言，zero-bug(没有错误)是我们的理想目标，good-enough(足够好)是我们的工作原则。

1. 足够好原则

good-enough 原则就是一种权衡投入/产出比的原则：不充分的测试是不负责任的；过

分的测试是一种资源的浪费，同样也是一种不负责任的表现。我们的操作困难在于：如何界定什么样的测试是不充分的，什么样的测试是过分的。目前状况唯一可用的答案是：制定最低测试通过标准和测试内容，然后具体问题具体分析。

2. 木桶原理和 80-20 原则

在确定软件测试原则时，我们一定要注意软件测试的规律——木桶原理和 80-20 原则。

1) 木桶原理

在软件产品的生产方面就是全面质量管理(TQM)的概念。产品质量的关键因素是分析、设计和实现，测试应该是融于其中的补充检查手段，其他管理、支持甚至文化因素也会影响最终产品的质量。应该说，测试是提高产品质量的必要条件，也是提高产品质量最直接、最快捷的手段，但不是一种根本手段。反过来说，如果将提高产品质量的砝码全部押在测试上，将是一场恐怖而漫长的灾难。

2) bug 的 80-20 原则

一般情况下，在分析、设计、实现阶段的复审和测试工作能够发现和避免 80%的 bug，而系统测试又能找出其余 bug 中的 80%，最后 4%的 bug 可能只有在用户的大范围、长时间使用后才会暴露出来。因为测试只能够保证尽可能多地发现错误，无法保证能够发现所有的错误。

3. 软件测试的一般原则

1) 测试应该基于用户需求，应该基于质量第一的事项去开展工作

依照用户的要求、配置环境和使用习惯进行测试并评价结果。软件测试只能证明软件中存在错误，而不能表明软件中没有错误。软件测试所起的作用也就是用于确定程序中缺陷的存在并帮助人们判断程序在实际中是否可用，而软件或程序是否可用必须依据事先定义好的产品质量标准。

2) 项目一启动，软件测试就开始，而不是等程序写完才开始测试

应该尽早开始测试，如尽早制定测试计划、尽早进行测试设计，以及测试从模块级开始等。其中，测试设计是关键，因为测试时间和资源是有限的，测试到所有情况是不可能的，但要避免冗余的测试；另外，软件测试中最困难的问题之一是何时停止测试，何时完成测试任务。

3) 充分覆盖程序逻辑，第三方测试会更有效、更客观

在软件测试中最忌讳的是程序编写者测试自己编写的程序，因为他们很难以批判者的姿态对自己编写的程序挑毛病。另外，他们的逻辑思维已定型，难以发现逻辑上的问题。

4. 基于经验的软件测试具体原则

我们在总结软件测试经验的基础上给出如下软件测试的具体原则：

1) 测试工作可以发现和显示软件缺陷的存在，但不能证明软件或系统不存在缺陷

测试可以减少软件中存在缺陷的可能性，但即使测试没有发现任何缺陷，也不能证明软件或系统是完全正确的，或者说是不存在缺陷的。

2) 测试的尽早介入

根据统计表明，在软件开发生命周期早期引入的错误占软件过程中出现的所有错误(包括最终的缺陷)数量的50%~60%。此外，研究结果表明，缺陷存在放大趋势。如需求阶段的一个错误可能会导致 N 个设计错误，因此，越是测试后期，为修复缺陷所付出的代价就会越大。因此，软件测试人员要尽早地且不断地进行软件测试，以提高软件质量，降低软件开发成本。

3) 测试活动要有组织、有计划、有选择

把尽早地和不断地进行软件测试作为软件开发者和软件测试者的座右铭；同时要明确测试工作量：穷举测试是不可能的，测试太少是不负责任，测试过多是一种浪费。因此，测试中有计划的活动能够大大地提高测试效率。

4) 选择最佳的测试策略

100%的测试是不可能的，不同的用户采用的测试策略是不同的，但我们一定要考虑软件测试的多、快、好、省。

所谓多、快、好、省，就是：①能够找到尽可能多的以至于所有的 bug；②能够尽可能早地发现最严重的 bug；③找到的 bug 是关键的、用户最关心的，找到 bug 后能够重现找到的 bug，并为修正 bug 提供尽可能多的信息；④能够用最少的时间、人力和资源发现 bug，测试的过程和数据可以重用。

5) 注重测试设计

测试设计决定了测试的有效性和效率，测试工具只能提高测试效率，测试用例设计则是软件测试的关键。

测试用例应由测试输入数据和对应的预期输出结果组成。设计测试用例时，应包括合理的输入条件和不合理的输入条件。一个好的测试用例应当是一个对以前未被发现的缺陷有高发现率的用例，而不是一个表明程序工作正常的用例。

另外，测试用例需要经常评审和修改，不断增加新的不同的测试用例来测试软件或系统的不同部分，保证测试用例永远是最新的，即包含着最后一次程序代码或说明文档的更新信息。这样软件中未被测试过的部分或者先前没有被使用过的输入组合就会重新执行，从而发现更多的缺陷。

由于软件测试的不成熟性和艺术性，我们在软件测试中不要放弃随机测试的方法。

6) 严格执行测试计划

测试中，一定要严格执行测试计划，坚决排除测试的随意性，必须对每一个测试结果做全面检查。

在测试中，程序员应避免检查自己的程序，否则很难发现思路错误和环境错误，或因心理因素导致测试可能不够彻底和全面。

测试人员要充分注意测试中因被测程序的编码规范、需求理解、技术能力、内部耦合性等导致错误扎堆这种虫子窝现象。一般测试后程序中残存的错误数目与程序中已发现的错误数目成正比，当一个软件被测出的错误数目增加时，更多的未被发现的错误存在的概率也随之增加。

另外，测试前必须明确预期的输出结果，否则实际的输出结果很可能成为检验的标准，测试失去意义。同时，作为专业的测试人员，要具有探索性思维和逆向思维，而不仅仅是做输出与期望结果的比较。

7) 测试活动依赖于测试内容

项目测试相关的活动依赖于测试对象的内容。对于每个软件系统，比如测试策略、测试技术、测试工具、测试阶段以及测试出口准则等的选择，都是不一样的。同时，测试活动必须与应用程序的运行环境和使用中可能存在的风险相关联。因此，没有两个系统可以以完全相同的方式进行测试。比如，对关注安全的电子商务系统进行测试，与一般的商业软件测试的重点是不一样的，它更多关注的是安全测试和性能测试。

8) 没有失效不代表系统是可用的

系统的质量特征不仅仅是功能性要求，还包括很多其他方面的要求，比如稳定性、可用性、兼容性等。假如系统无法使用，或者系统不能完成客户的需求和期望，那么这个系统的研发是失败的。同时在系统中发现和修改缺陷也是没有任何意义的。

在开发过程中用户的早期介入和接触原型系统就是为了避免这类问题的预防性措施。有时候，可能产品的测试结果非常完美，可最终的客户并不买账，因为这个开发角度完美的产品可能并不是客户真正想要的产品。

9) 测试的标准是用户的需求

提供软件的目的是帮助用户完成预定的任务，并满足用户的需求。这里的用户并不特指最终软件测试使用者。比如我们可以认为系统测试人员是系统需求分析和设计的客户。软件测试的最重要目的之一是发现缺陷，因此测试人员应该在不同的测试阶段站在不同用户的角度去看问题，系统中最严重的问题是那些无法满足用户需求的错误。

10) 尽早定义产品的质量标准

只有建立了质量标准，才能根据测试的结果对产品的质量进行分析和评估。同样，测试用例应该确定期望的输出结果。如果无法确定测试期望结果，则无法进行检验。必须用预先精确对应的输入数据和输出结果来对照检查当前的输出结果是否正确，做到有的放矢。

11) 测试贯穿于整个生命周期

由于软件的复杂性和抽象性，在软件生命周期的各个阶段都可能产生错误，测试的准备和设计必须在编码之前就开始，同时为了保证最终的质量，必须在开发过程的每个阶段都保证其过程产品的质量。因此不应当把软件测试仅仅看作软件开发完成后的一个独立阶段的工作，应当将测试贯穿于整个生命周期始末。

软件项目一旦启动，软件测试就应该介入，而不是等到软件开发完成。在项目启动后，测试人员在每个阶段都应该参与相应的活动。或者说在每个开发阶段，测试都应该对本阶段的输出进行检查和验证。比如在需求阶段，测试人员需要参与需求文档的评审。

12) 第三方或独立的测试团队

由于心理因素，人们潜意识里都不希望找到自己的错误。基于这种思维定式，人们难以发现自己的错误。因此，由严格的独立测试部门或第三方测试机构进行软件测试将更客观、公正，测试活动也会达到更好的效果。

但是，第三方或独立的测试团队这条原则，并不是说所有的测试完全由他们来完成。一定程度的独立测试(可以避免开发人员对自己代码的偏爱)，可以更加高效地发现软件缺陷。但独立测试不是完全的替代物，因为开发人员也可以高效地在他们的代码中找出很多缺陷。在软件开发的早期，开发人员对自己的工作产品认真地进行测试，这也是开发人员的职责之一。

13) 妥善保存测试计划、测试用例、出错统计和最终分析报告

通过配置管理、测试管理等技术手段，保存测试计划、测试用例、出错统计和最终分析报告，除了方便维护及结果重现外，对于组织和团队的建设、工作的改进、成果的积累、资源的重用等有重大好处。

2.1.5　软件测试质量度量

软件测试是软件质量控制的重要方式和重要手段，但软件测试本身的质量又该如何度量？尽管有关这个问题目前还没有权威的结论，但也有一些共识，如软件测试度量的难度在于不能直接从软件产品的质量反映软件测试的效果。对于软件测试的度量，应该从对软件产品的度量转移到对软件测试产出物的度量，以及对测试过程的度量。

软件测试度量的目的是改进软件测试的质量，提高测试效率，改进测试过程的有效性。开展软件测试质量度量，最关键的一项工作就是对软件测试人员工作质量的度量。这是因为测试人员是测试过程的核心人物，测试人员的工作质量会极大地影响测试的质量以及产品的质量。对测试人员的工作做出评价一般由测试经理或项目经理、质量保证人员以及开发人员这三类人员进行综合考核或评判。除了人员考核这个管理内容外，图 2-1 所示的表格给出了软件开发与软件测试质量度量的相关指标及度量范围。

指标名称	定义	度量范围
工作量偏差	（（实际工作量-计划工作量）/计划工作量）*100%	进度
测试执行率	（实际执行的测试用例数/测试用例总数）*100%	测试进度
测试通过率	（执行通过的测试用例数/测试用例总数）*100%	开发质量
需求（测试用例）覆盖率	（已设计测试用例的需求数/需求总数）*100%	测试设计质量
需求通过率	（已测试通过的需求数/需求总数）*100%	进度
测试用例命中率	（缺陷总数/测试用例数）*100%	测试用例质量
二次故障率	（Reopen的缺陷/缺陷总数）*100%	开发质量
NG率	（验证不通过的缺陷/缺陷总数）*100%	开发质量
缺陷有效率	（无效的缺陷/缺陷总数）*100%	测试
缺陷修复率	（已解决的缺陷/缺陷总数）*100%	开发
缺陷生存周期	缺陷从提交到关闭的平均时间	开发、测试
缺陷修复的平均时长	缺陷从提交到修复的平均时间	开发
缺陷关闭的平均时长	缺陷从修复到关闭的平均时间	测试
缺陷探测率	（测试者发现的缺陷数/（测试者发现的缺陷+客户发现的缺陷））*100%	测试质量

图 2-1　软件测试的质量度量

2.1.6　软件测试与软件开发各阶段的关系

软件开发是一个自顶向下、逐步细化的过程。软件计划阶段定义软件作用域；软件需求分析阶段建立软件信息域、功能和性能需求、约束等；软件设计阶段把设计用某种程序设计语言转换成程序代码。

测试是依相反顺序自底向上、逐步集成的过程。对每个程序模块进行单元测试，消除程序模块内部逻辑和功能上的错误和缺陷；对照软件设计进行集成测试，检测和排除子系统或系统结构上的错误；对照需求，进行确认测试；最后从系统整体出发，运行系统，看是否满足要求。软件测试与软件开发各阶段的关系参见图 2-2。

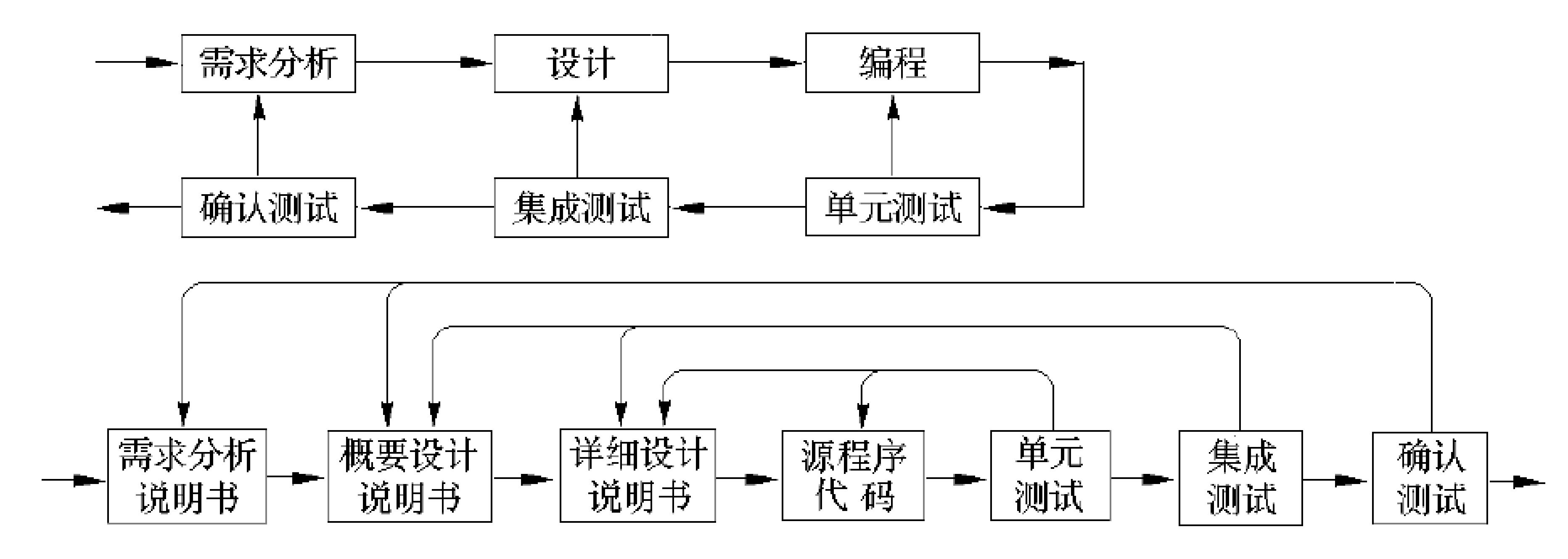


图 2-2　软件测试与软件开发各阶段的关系图

软件测试是用人工或自动的方法执行软件并把观察到的行为特性与所期望的行为特性进行比较的过程。按照传统的观点，软件测试是软件开发过程中的一项活动。随着对软件测试方法、测试工具和测试技术的研究，测试的概念已经从编程后的评估过程发展成软件开发生命周期中每个阶段的一项必需的活动。软件开发对应的软件测试过程如图 2-3 所示。

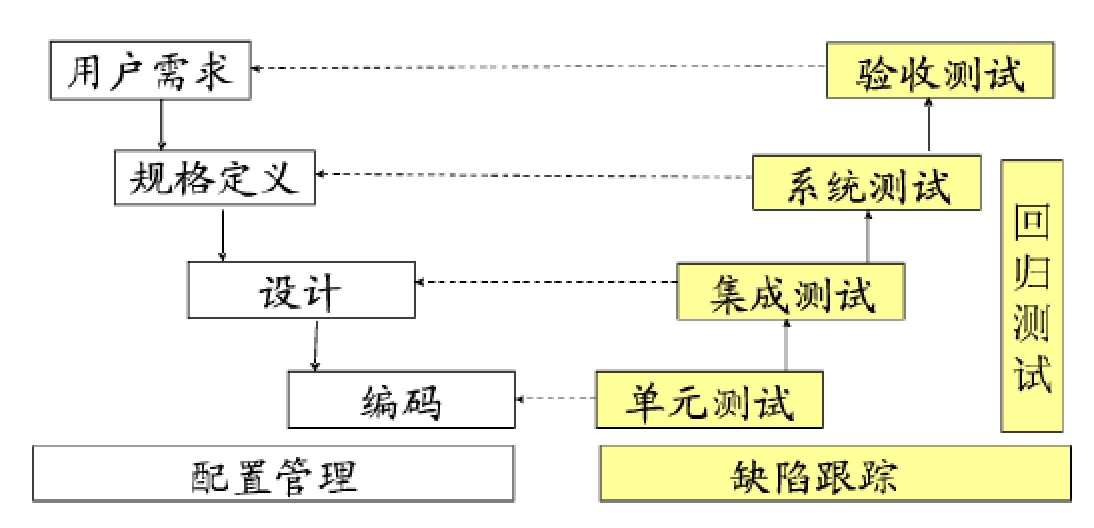


图 2-3　软件开发对应的软件测试过程

2.2　软件测试工作

软件开发工作的根本是尽量实现软件用户的需求，软件测试工作的根本是检验软件系统是否满足软件用户的需求。因此，我们可以说：软件测试的主要工作内容是验证和确认软件的设计、开发是否符合需求；验证是保证软件正确地实现了客户的业务功能，即保证软件实现了需求所规定的内容；确认是通过使用已开发好的系统体验各项流程，目的是想证实在给定的外部环境中软件的逻辑是否正确。

不论是软件模块还是整个系统，它们有着共同的测试内容，如正确性测试、容错性测

试、性能与效率测试、易用性测试、文档测试等。常用的测试技术或手段是白盒测试及黑盒测试。

软件测试工作涉及多方面的内容和因素，但最为重要的是：测试工作的流程、测试工具的支持以及测试人员的素质这三项内容。

2.2.1　软件测试工作的流程

软件测试工作贯穿整个软件开发周期，从需求提出到产品发布，测试人员几乎要全程参与其中，并按照特定的测试工作流程和企业对软件的质量要求开展测试工作。

1. 测试工作的基本流程

测试工作的基本流程如下：

1) 需求阅读与评审

需求文档由 SE(System Engineer)根据客户需求编制，完成后提交到软件配置管理中的开发库。测试人员从开发库中下载该文档阅读，在阅读过程中要记录需求中模糊的部分，提出测试环境方案并绘制测试系统结构图，还要根据现有的软硬件条件对项目的可测性进行评估。

接下来是参加由 SE 发起的需求评审会议，在会议上测试人员要做的工作包括：从 SE 处获取需求文档中模糊部分的详细解释；向与会人员介绍测试环境方案，与会人员进行评审，评审完成后确定最终的方案；如果现有的条件不足以保证软件的各个模块都可测，需要就不可测的部分与参会人员探讨，最终提出一个解决方案，硬件不满足的购置设备，软件不满足的编制模拟程序。

2) 用例设计与评审

在这个过程中，不要求每个用例都非常详细，但是要求用例要全面并充分考虑异常，保证各个模块都能够得到充分测试。

3) 环境搭建

根据之前评审确定的测试环境方案搭建测试环境。

4) 软件测试

从开发库中获取软件开发人员提交的软件测试版本，根据需求文档和之前设计的测试用例进行测试。如果发现问题，用缺陷管理工具提交问题单，问题单提交后，相关的开发人员会收到邮件提醒并解决问题，问题处理完毕后重新提交版本，测试人员重新获取版本并验证问题的解决情况，循环上述过程，直到整个软件测试完毕。

在此过程中，还需要细化测试用例，详细描述每一个用例的步骤。此外，每天工作结束后要向测试经理汇报测试进度，包括发现了哪些 bug 以及每个 bug 的处理情况(已提交问题单、开发已处理完毕、回归测试中、回归测试完毕)。

5) 编写有关测试文档(测试用例设计、测试报告、问题报告等)

测试用例可使用相关工具导出，形成 Word 文档；测试报告的内容包括测试环境结构图、测试环境介绍、测试方法、测试结论等。

6) SE 与测试经理审核

SE 和测试经理根据测试报告对测试工作进行审核，审核通过后测试工作完成。

2. 企业对软件质量的要求是测试工作开展的基础

在开展测试工作之前，企业必须做到对软件质量水平有明确的定位，坚持质量第一的原则，以客户满意为最终目标，要求测试人员发现任何问题都要及时反馈。同时规定：与需求的符合程度是衡量软件质量的唯一标准。以上两点使得测试人员在发现问题时有明确的处理原则和参照文档，避免了测试的随意性。这样，测试人员就能够充分地发挥他们的主观能动性，高质量地开展测试工作。比如：

(1) 测试人员在每个项目测试之初，认真地规划好时间点。从测试方案设计到环境搭建，再到版本测试，都有较为合理和详细的时间规定，保证测试在时间上的可控性。同时采用每天上报的方法，对测试进度进行跟踪，这对保证版本按时测试完毕有重要意义。

(2) 测试方案是否可行、是否能够对软件进行有效测试、缺少哪些软硬件设备，这些都需要进行充分论证，若有问题及时解决，直到整个方案符合要求。

(3) 在评审测试用例时，对测试人员设计的用例进行讨论，补充遗漏，提示重点，特别是在异常的考虑上，能够集思广益，使得最终敲定的测试用例有良好的质量保证。

(4) 测试人员注重自身专业技能的提高，保证测试工作的质量和效率。

2.2.2　软件测试工具对测试工作的支持

进入 20 世纪 90 年代，软件行业开始迅猛发展，软件的规模变得非常大，在一些大型软件开发过程中，测试活动需要花费大量的时间和成本，而当时的测试手段几乎完全都是手工测试，测试的效率非常低；并且随着软件复杂度的提高，出现了很多通过手工方式无法完成测试的情况，尽管在一些大型软件的开发过程中，人们尝试编写一些小程序来辅助测试，但这还是不能满足大多数软件项目的统一需要。于是，很多测试实践者开始尝试开发商业的测试工具来支持测试，辅助测试人员完成某一类型或某一领域内的测试工作，测试工具逐渐盛行起来。人们普遍意识到，工具不仅仅是有用的，而且要对今天的软件系统进行充分的测试，工具是必不可少的。

事实上，在繁重的测试工作中，测试人员使用软件测试工具有如下好处：

1. 提高工作效率

软件测试的工作量很大(据统计，测试会占用 40%的开发时间；一些可靠性要求非常高的软件，甚至会占到 60%)；另外，测试中的许多操作是重复的、非智力性的和非创造性的，但要求准确、细致。最后，那些固定的、重复性的工作，可以由测试工具来完成，这样就使得测试人员能有更多的时间来计划测试过程，设计测试用例，使测试更加完善。

2. 保证测试的准确性

手工测试经常会犯一些人为错误。为此需要投入大量的时间和精力；而工具的特点是能保证测试的准确性，防止人为疏忽造成的错误。

测试工具可以进行部分的测试设计、实现、执行和比较工作，这些工作中的很多可应用自动化技术来完成。比如采用自动比较技术，可以自动完成对测试用例执行结果的判断，

从而避免人工比对存在的疏漏问题。设计良好的自动化测试，在某些情况下可以实现夜间测试和无人测试。在大多数情况下，软件测试自动化可以减少开支，增加有限时间内可执行的测试，在执行相同数量的测试时节约测试时间。

3. 有些测试很难开展，必须使用工具(如性能测试等)

测试工具可以执行一些手工难以执行或是无法执行的测试。这是因为软件测试工作相当复杂，要求非常严格，很多测试在手工测试环境是无法完成的。

4. 测试工具很好地保证测试工作的规范性和一致性

软件工程最重要的内容就是管理，软件测试同样也是将管理放在第一位。

5. 测试工具体现了先进的测试思想、方法和技术

测试工具的使用能够快速地提升软件测试的专业化水平。因此，测试工具的选择和推广也越来越受到重视。

在软件测试工具平台方面，商业化的软件测试工具已有很多，如捕获/回放工具、Web 测试工具、性能测试工具、测试管理工具、代码测试工具等，这些都有严格的版权限制且价格较为昂贵，由于受到价格和版权的限制而无法自由使用。当然，一些软件测试工具开发商对于某些测试工具提供了 Beta 测试版本以供用户使用有限次数。幸运的是，在开放源码社区中也出现了许多软件测试工具，它们已得到广泛应用且相当成熟和完善。

2.2.3　软件测试工作的几个认识误区

随着软件规模的不断扩大，软件设计的复杂程度不断提高，软件开发中出现错误或缺陷的机会越来越多。同时，市场对软件质量重要性的认识逐渐增强。所以，软件测试在软件项目实施过程中的重要性日益突出。但现实情况是，与软件编程比较，软件测试的地位和作用还没有真正受到重视，对于很多人(甚至是软件项目组的技术人员)还存在软件测试的认识误区，这进一步影响了软件测试活动的开展和软件测试质量的真正提高。

1. 整体认识上重开发而轻测试

整个软件项目制作过程中的各种活动按重要程度进行排序，结果软件测试几乎从未名列前茅过。另外，如果在软件项目面临着时间压力、必须加快研制速度的情况下，通常软件测试会成为换取项目进度的牺牲品。重开发而轻测试在软件开发中颇为普遍。

2. 软件开发完成后进行软件测试

人们一般认为，软件项目要经过以下几个阶段：需求分析、概要设计、详细设计、软件编码、软件测试、软件发布。据此，认为软件测试只是软件编码后的一个过程。这是不了解软件测试周期的错误认识。

软件测试是一系列活动，包括软件测试需求分析、测试计划设计、测试用例设计、执行测试。因此，软件测试贯穿于软件项目的整个生命周期。在软件项目的每一个阶段都要进行不同目的和内容的测试活动，以保证各个阶段的正确性。软件测试的对象不仅仅是软件代码，还包括软件需求文档和设计文档。软件开发与软件测试应该是交互进行的，例如，单元编码需要单元测试、模块组合阶段需要集成测试。如果等到软件编码结束后才进行测

试，那么测试的时间将会很短，测试的覆盖面将很不全面，测试的效果也将大打折扣。更严重的是，如果此时发现了软件需求阶段或概要设计阶段的错误，要修复该类错误，将会耗费大量的时间和人力。

3. 软件测试是为了证明软件的正确性

测试的目的是证伪而不是证真。事实上，要证明程序的正确性是不可能的，一个大型的集成化的软件系统不能被穷尽测试以遍历其每条路径，即使遍历了所有的路径，错误也仍有可能隐藏。测试只是为了尽可能地发现错误。

4. 软件发布后如果发现质量问题，那是软件测试人员的错

这种认识严重挫伤了软件测试人员的积极性。软件中的错误可能来自软件项目中的各个过程，软件测试只能确认软件存在错误，不能保证软件没有错误，因为从根本上讲，软件测试不可能发现全部的错误。从软件开发的角度看，软件的高质量不是软件测试人员测出来的，是靠软件生命周期的各个过程设计出来的。出现软件错误，不能简单地归结为某个人的责任，有些错误的产生可能不是技术原因，可能来自混乱的项目管理。应该分析软件项目的各个过程，从过程改进方面寻找产生错误的原因和改进的措施。

5. 软件测试要求不高，随便找个人就行

很多人都认为软件测试就是安装和运行程序，点点鼠标，按按键盘的工作。这是由于不了解软件测试的具体技术和方法造成的。测试人员能力的高低会极大地影响测试工作的成败，测试者必须既明白被测软件系统的概念，又要会使用工程中的那些工具，要做到这一点还需要有几年以上的编程经验。随着软件工程学的发展和软件项目管理经验的提高，软件测试已经形成一门独立的技术学科，演变成一个具有巨大市场需求的行业。软件测试技术不断更新和完善，新工具、新流程、新测试设计方法都在不断更新，需要掌握和学习很多测试知识。所以，具有编程经验的程序员不一定是一名优秀的测试工程师。软件测试包括测试技术和管理两个方面，完全掌握这两个方面的内容，需要很多测试实践经验和不断学习的精神。

6. 软件测试是软件开发的对头

测试人员和开发人员经常无法有效地一起工作。其实，开发和测试作为一个整体都是服务于产品，都要为产品的质量负责。因此测试应该是开发的朋友，而不是开发的对头。

7. 软件测试是测试人员的事情，与程序员无关

开发和测试是相辅相成的过程，需要软件测试人员、程序员和系统分析师等保持密切的联系，需要更多的交流和协调，以便提高测试效率。另外，对于单元测试主要应该由程序员完成，必要时测试人员可以帮助设计测试用例。对于测试中发现的软件错误，很多需要程序员修改代码才能修复。程序员可以通过有目的地分析软件错误的类型、数量，找出产生错误的位置和原因，以便在今后的编程中避免同样的错误，积累编程经验，提高编程能力。

8. 项目进度吃紧时少做些测试，时间富裕时多做些测试

这是不重视软件测试的表现，也是软件项目过程管理混乱的表现，必然会降低软件测

试的质量。软件项目的顺利实现需要有合理的项目进度计划，其中包括合理的测试计划，对项目实施过程中的任何问题都要有风险分析和相应的对策，不要因为开发进度的延期而简单地缩短测试时间、减少人力和资源。因为缩短测试时间带来的测试不完整，对项目质量的下降引起的潜在风险往往造成更大的浪费。克服这种现象的最好办法是加强软件过程的计划和控制，包括软件测试计划、测试设计、测试执行、测试度量和测试控制。

9. 软件测试是没有前途的工作，只有程序员才是软件高手

由于我国软件整体开发能力比较低，软件过程很不规范，很多软件项目的开发都还停留在小作坊和垒鸡窝阶段。项目的成功往往靠个别全能程序员决定，他们负责总体设计和程序详细设计，认为软件开发就是编写代码，给人的印象往往是程序员是真正的牛人，具有很高的地位和待遇。因此，在这种环境下，软件测试很不受重视，软件测试人员的地位和待遇自然就很低了，甚至软件测试变得可有可无。随着市场对软件质量要求的不断提高，软件测试正变得越来越重要，相应的软件测试人员的地位和待遇将会逐渐提高。在微软等软件开发过程比较规范的大公司，软件测试人员的数量和待遇与程序员没有多大差别，优秀测试人员的待遇甚至比程序员还要高。软件测试将会成为一个具有很大发展前景的行业，软件测试大有前途，市场需要更多具有丰富测试技术和管理经验的测试人员，他们同样是软件专家。

10. 软件测试就是程序测试，测试发现了错误就说明是程序员所编写的程序有问题

在软件项目开发的整个过程中，前一阶段工作中发生的问题如未及时解决，很自然地要影响到下一阶段。表现在程序中的缺陷可能因设计阶段甚至需求分析阶段的问题所致，大多数软件缺陷并非来自编码过程，从小项目到大项目基本上都是如此。即使是针对源程序测试所发现的故障，其根源也可能存在于软件开发前期的各个阶段。因此不能简单地把程序中的错误全都归罪于程序员。

11. 期望用测试自动化代替大部分人工劳动

目前，很多企业首先是从节约成本的角度考虑引入测试自动化工具的。自动化测试工具的确能用于完成部分重复、枯燥的手工作业，但不要指望用来代替人工测试。一般来讲，产品化的软件更适于功能测试的自动化，由标准模块组装的系统更好，因为其功能稳定，界面变化不大。性能测试似乎更加依赖于自动化测试工具(如模拟多个虚拟用户和收集性能指标等)，但考虑购买时也要注意，假设一个软件不支持.NET，也许它就不一定适用于贵公司。

12. 所有软件缺陷都可以修复

在软件测试中，有一种令人沮丧的现实是，即使拼尽全力，也不是所有的软件缺陷都能修复。但是，这并不意味着软件测试未达到目的，或者项目小组将发布质量欠佳的产品。事实上，也不需要对所有的软件缺陷进行修复。

13. 认为软件测试文档不重要

对于软件测试，四个最典型的书面文档是测试计划、测试用例、缺陷列表、测试报告。

(1) 测试计划：测试作为整个项目工程的一部分，在早期做出较为详细的测试范围、

人力预算、执行时间、技术需求/培训和软硬件资源占用等方面的考虑，便于有目的、有计划地完成后面的测试工作，测试团队也能更好地与其他团队协作。后期评审中，以此为基线，更容易发现执行中的问题和及时做出调整。

(2) 测试用例：可以让参与测试的人员，花足够的精力，第一时间去系统地理解需求。在此基础上进行的同行评审，则可以防止需求理解得不彻底和偏差，尽早发现可能存在的测试漏洞。同时，测试用例作为新旧人员交替时知识传递的媒体，有利于新人(时常也有从其他团队借调的人员)尽快进入特定模块的测试工作，而不是被成堆的需求文档所淹没。

(3) 缺陷列表：好的缺陷管理应该引入一个软件系统，而不是使用传统意义上的 Word 或 Excel 文档。想象一下，当缺陷数目达到几百个时，按照某些工业模板去生成 bug 文件，怎能做到方便地管理、查询和分析？如果测试周期持续很长，测试人员又不止一个，报 bug 前寻找雷同的旧记录都将非常费时、费力。

(4) 测试报告：把测试的过程和结果写成文档，并对发现的问题和缺陷进行分析，为纠正软件存在的质量问题提供依据，同时为软件验收和交付打下基础。另外，测试报告是测试阶段最后的文档产出物，一份详细的测试报告包括产品质量和测试过程的评价。测试报告基于测试中的数据采集以及对最终测试结果的分析，比如覆盖率分析、缺陷分析等。

14. 期望短期通过增加软件测试投入，迅速达到零缺陷率

即使我们有充裕的资金，也不是说软件测试投入得越多越好。增加测试人力和时间上的投入，的确能帮助我们找出更多的缺陷。但二者不是一种线性关系，随着测试投入的不断放大，产品质量上升是逐渐收敛的。一个项目投入 10 名测试人员，发现了 70%的缺陷，并不表明投入 20 个人就能找出几乎所有的缺陷，也许这个数字只会是 85%。

所以，测试经理要根据公司的具体情况，如策略方针、市场定位以及产品类别等因素，来决定开发和测试人员的比率和测试投入，这才是合理的。举例来说，做一个 ERP 解决方案、一个杀毒软件乃至一个航天飞船的控制系统，对测试的要求(包括技术结构和资源占用等)肯定是不一样的。

另外，软件质量的好坏和整个软件开发过程的优劣是密不可分的，不是仅仅通过加强测试就可以完全控制的。

15. 规范化软件测试会增加项目成本

增加软件测试人员和预留项目测试时间，表面上看增加了人员成本或延长了项目周期，并为此会投入更多的项目资金。然而，越早发现软件中存在的问题，开发费用就越低；美国质量保证研究所对软件测试的研究结果表明：在编码后修改软件缺陷的成本是编码前的 10 倍，在产品交付后修改软件缺陷的成本是交付前的 10 倍；软件质量越高，软件发布后的维护费用越低。

所以，即使我们的客户有最好的忍耐力，或愿意免费充当我们的测试队伍，前期找出缺陷也会省去很多的后期排错和维护费用。加上由于产品不成熟而造成市场声誉受损，单从经济效益方面来考虑，前期足够的测试也是值得投入的。

2.3　软件测试职业

由于近年来我国软件行业的产业升级以及软件开发模式的升级(软件开发的单打独斗升级为工业化、流水线式的生产模式)，作为工业化的产品，软件测试也就成为软件开发企业必不可少的质量监控环节，贯穿于软件产品研发周期内的每一个环节中，在整个软件开发的系统工程中占据相当大的比重。在软件产业发达国家，软件企业一般把 40%的工作花在测试上，测试人员和开发人员之比平均在 1:1 以上，软件测试费用占整体开发费用的 30%~50%，对于高可靠、高安全的软件，测试费用则相当于整个软件项目开发所有费用的 3 至 5 倍。这些说明了软件测试的重要性。而目前我国无论是政府、企业还是高等院校，对软件测试工作和人才培养一直不够重视，大家重开发、轻测试，以至于我国在软件测试上的投入远远低于在软件开发上的投入，远远低于软件产业发达国家在软件测试上的投入。软件测试人才的培养数量较发达国家以及我国产业升级相当滞后，这就形成了软件测试人才的供给远小于需求的现状。

最近几年我国的媒体时常报道：中国软件测试人才缺口 30 万，软件测试工程师将成为未来 10 年最紧缺的人才之一，众多国内外优秀企业对高端测试人才以年薪 10 万、20 万、30 万进行招聘。由此可见，软件测试目前是一个朝阳行业。但是，近几年在北京、上海、深圳举办的各种大型招聘会上，多家企业纷纷打出各类高薪招聘软件测试人员的海报，出人意料的是，收到的简历尚不足招聘岗位数的 50%，而合格的竟不足 30%。这说明我们的软件测试从业人员还有很大一部分不满足当今社会的需求，高素质、专业化的软件测试人才非常紧缺，具有一定测试经验的软件测试工程师很受市场青睐，供不应求。而另一层含义是，我们还有很大的提升空间。

目前，软件测试工作越来越得到足够重视，现在测试人员的待遇和开发人员的待遇非常接近。

2.3.1　软件测试职业发展

软件测试职业发展方向与其他职业发展方向是相吻合的，也可以分为管理路线、技术路线、管理+技术路线。

软件测试人员的职业发展是基于软件测试团队这个组织的人员角色转变或升迁的。也就是说，软件测试人员要确立在软件测试团队这个组织中的职业规划图(即个人 Roadmap)，然后基于这个 Roadmap 确立各个角色的职责以及各角色之间的相互联系和发展顺序。这样，软件测试人员在测试团队中的发展目标也就确立了。

1. 测试团队的基本构成

作为一支测试团队或一支比较健全的测试团队，应该具有下面这些人员角色：

(1) 测试经理：主要负责人员的招聘、培训和管理，以及资源调配、测试方法改进等。

(2) 实验室管理人员：设置、配置和维护实验室的测试环境，如服务器和网络环境等。

(3) 测试配置管理人员：审查流程，并提出改进流程的建议；建立测试文档所需的各种模板，进行测试的配置管理，检查软件缺陷描述及其他测试报告的质量等。

(4) 测试组长：业务专家，负责项目的管理、测试计划的制定、项目文档的审查、测试用例的设计和审查、任务的安排、与项目经理及开发组长的沟通等。

(5) 一般(初级)测试工程师：编写、执行测试用例，编制有关的测试文档或开展其他相关的测试任务。

对于比较大规模的测试团队，测试工程师分为三个层次：初级测试工程师、测试工程师、资深(高级)测试工程师等，同时还设立自动化测试工程师、系统测试工程师和架构工程师。

对于规模很小的测试小组，可能没有设置测试经理，只有测试组长。这时测试组长承担测试经理的部分责任，如参加面试工作、资源管理、团队发展等，并且要做内审员的工作，检查软件缺陷描述及其他测试报告的质量等。资深测试工程师不仅要负责设计规格说明书的审查、测试用例的设计等，还要设置测试环境，即承担实验室管理人员的责任。对于一些大型的企业或专业的测试团队，除了测试经理外，在他之上还有测试总监。测试总监是在公司或企业层面上对整个测试工作进行管理，包括行政上和技术上的管理。

2. 测试人员的职位及责任

基于前面测试人员角色的划分，对应其职位，从一般(初级)测试工程师开始，再到资深测试工程师，最后到测试经理，自底向上地了解他们的责任。测试工程师虽然和初级测试工程师责任不一样，但测试工程师肯定能做好所有要求初级测试工程师做好的工作。

不同层次的测试工程师，责任也有一定的区别，但都是技术工作，主要任务是设计和执行各种测试任务，是测试工作的基础。

1) 初级测试工程师

初级测试工程师的责任比较简单，还不具备完全独立的工作能力，需要测试工程师或资深测试工程师的指导，要求比较低。初级测试工程师的主要责任：

① 了解和熟悉产品的功能、特性等；
② 验证产品在功能、界面上是否和产品规格说明书一致；
③ 按照要求，执行测试用例，进行功能测试、验收测试等，并能发现所暴露的问题；
④ 清楚地描述所出现的软件问题；
⑤ 努力学习新技术和软件工程方法，不断提高自己的专业水平；
⑥ 使用简单的测试工具；
⑦ 接受测试工程师的指导，执行主管交代的其他工作。

2) 测试工程师

其适用范围是入行软件测试 3 年内的常规测试从业者。测试工程师的责任相对多些，具有独立的工作能力，其主要工作内容是按照测试主管或测试组长(即直接上司)分配的任务计划，熟悉测试流程、测试方法和技术，编写测试用例、执行测试用例(包括参与自动化测试)、提交软件缺陷，包括提交阶段性测试报告、参与阶段性评审等。

测试工程师以执行测试为主，其主要责任是：

① 熟悉产品的功能、特性，审查产品规格说明书；
② 根据需求文档或设计文档，可以设计功能方面的测试用例；

③ 根据测试用例，执行各种测试，发现所暴露的问题；
④ 全面使用测试工具，包括测试脚本的编写；
⑤ 安装、设置简单的系统测试环境；
⑥ 报告所发现的软件缺陷，审查软件缺陷，跟踪缺陷修改的情况，直到缺陷关闭；
⑦ 写测试报告；
⑧ 负责对初级测试工程师进行指导，执行主管交代的其他工作。

3) 资深测试工程师

其适用范围是具有 3~5 年职业经验的测试从业者。资深测试工程师不仅具有良好的技术、产品分析能力、解决问题能力、丰富的测试工作经验，而且有较好的编程、自动化测试经验，熟悉测试流程、测试方法和技术，能够解决测试工作中可能遇到的各种技术问题。

资深测试工程师的工作内容是根据项目经理或测试经理的计划安排，调配测试工程师执行模块级或项目级测试工作，并控制与监督软件缺陷的追踪，保证每个测试环节与阶段的顺利进行。严格来说，这个级别更多属于测试的设计者，因为企业的测试流程搭建是由更高级别的测试经理或相关管理者来做的，资深测试工程师负责该流程的具体实施；而更多的工作，是思考如何对软件进行更加深入、全面的测试。资深测试工程师比较有创造性的工作内容就是测试设计，而恰恰很多公司忽略了或没有精力来执行此工作内容。应该说，在一个企业里做了 3 年左右测试工作的人员，很容易晋升到该职位，而之所以晋升，是与个人测试技术的过硬、测试方法的丰富，加上对测试流程的监控力与执行力的职业素质息息相关。

资深测试工程师的主要责任是：
① 负责系统一个或多个模块的测试工作；
② 制定某个模块或某个阶段的测试计划、测试策略；
③ 设计测试环境所需的系统或网络结构，安装、设置复杂的系统测试环境；
④ 熟悉产品的功能、特性，审查产品规格说明书，并提出改进要求；
⑤ 审查代码；
⑥ 验证产品是否满足规格说明书所描述的需求；
⑦ 根据需求文档或设计文档，设计复杂的测试用例；
⑧ 负责对测试工程师进行指导，执行主管交代的其他工作。

4) 测试实验室管理员

测试实验室管理员主要负责建立、设置和维护测试环境，保证测试环境的稳定运行，其主要责任是：
① 负责测试环境所需的网络规划和建设，维护网络的正常运行；
② 建立、设置和维护测试环境所需的应用服务器和软件平台；
③ 申请所需要的、新的硬件资源、软件资源，协助有关部门进行采购、验收；
④ 对使用实验室的硬件、软件资源的权限进行设计、设置，保证其安全性；
⑤ 安装新的测试平台、被测试的系统等；
⑥ 优化测试环境，提高测试环境中网络、服务器和其他设备的运行性能。

5) 测试配置管理人员

测试配置管理人员在 QA 工作中起着很重要的角色，负责测试产品的上载、打包和发布等测试配置管理工作，其主要责任是：

① 负责源程序代码管理系统的建立、管理和维护；

② 文件名定义规范，建立合理的程序文件结构和存储目录结构；

③ 为程序的编译、链接等软件包构造自动处理文件；

④ 保证测试最新的产品包并上传到相应的服务器上，并确认各模块或组件之间相互匹配；

⑤ 每天为各项目新的或修改的代码重新构造新的软件包；

⑥ 确保不含病毒，不缺图片和各种文件；

⑦ 文件包的接收、发送、存储和备份等。

6) 测试组长

测试组长一般要有 3 年以上测试经验，1 年以上的 2 至 7 人测试项目管理经验。具备资深测试工程师的能力和经验，熟练掌握全面的测试理论，能够很好地使用相关测试技术和工具，熟悉软件测试流程，具有优秀的测试文档编写能力，包括测试计划、测试方案、测试用例、测试报告等。测试组长可能在技术上相对弱些，不是小组内最强的，但他能够带领团队内的测试工程师，执行所负责软件的测试计划，跟踪并报告测试计划的执行进度。另外，测试组长要有较强的交流和沟通能力。测试组长的主要责任是：

① 负责测试项目启动的全面管理，包括测试小组的业务管理和人员管理；

② 与客户沟通交流，建立良好的合作氛围；

③ 制定测试计划，跟踪并报告测试计划的执行进度；

④ 执行测试并及时反馈项目状态，分析项目风险；

⑤ 负责或组织本项目组的培训工作；

⑥ 发展团队核心竞争力并扩展团队工作内容。

7) 测试经理

测试经理是更高级别的测试管理者。对于大中型软件公司，该职位尤为重要，并且对其职业要求也比较高，一般适合 4~8 年的测试从业者，在管理与技术能力双双比较成熟的情况下，可以结合具体环境晋升到该级别。

测试经理的主要工作在团队、资源和项目等管理上，不同于测试组长。测试组长主要集中在项目管理上，一般不负责测试人员的招聘、流程定义等管理工作，而且偏重技术。测试经理对产品的质量负全面责任，有责任向公司最高管理层反映软件开发过程中的管理问题或产品中的质量问题，使公司能全面掌握生产和质量状况。

测试经理的主要职责是：

① 负责企业级或大型项目级总体测试工作的策划与实施。测试经理除了需要统筹整个企业级或项目级测试流程外，还要对不同软件架构、不同开发技术下的测试方法进行研究与探索，为企业的测试团队成员提供指导与解决思路，同时还要合理调配不同专项测试的人力资源(如业务测试工程师、自动化测试工程师、白盒测试工程师、性能测试工程师)，对软件进行全面的测试；

② 负责被测项目(测试/质量/开发)内的整个开发生命周期业务，包括项目成本分析、进度安排、计划和人员分工；

③ 负责与客户(甚至与开发团队)的交流与沟通；

④ 负责部分的销售性或技术支持性工作，比如为一些用户提供交流和演示。

8) 测试总监

测试总监属于常规发展路线的最高域，该职位一般在大型或跨国型软件企业，或者专向于测试服务型企业设立。一般设立测试总监的企业，该职位都相当于技术总监 CTO 或副总的级别，是企业级或集团级测试工作的最高领导者，驾驭着企业全部的测试和与测试相关资源，管理着企业的全部测试及质量类工作。而其职业要求，也是技术与管理双结合。

测试总监的主要职责是：

① 根据企业年度预算目标，制定本部门年度人员配置计划和费用预算；

② 负责项目测试计划的审核，并带领测试团队设计、执行、优化测试过程，丰富测试手段，引入新的测试框架和测试策略，持续改进软件测试过程，确保项目测试符合流程与规范；

③ 与其他测试人员、开发人员、项目管理人员沟通和协作，推动整个项目的顺利进行。同时组织、协调资源，确保本部门各项测试工作按计划完成；

④ 制定本部门的《年度绩效目标责任书》，并跟进绩效目标完成；

⑤ 负责本部门测试团队的建设和管理，包括测试团队中各类测试人员的培训、指导、培养等管理工作，不断提升公司的测试能力，带领和指导整个测试团队科学、规范、合理、高效地协同工作；

⑥ 维护测试流程，统计和分析测试结果，提高及改进测试效率和质量；

⑦ 负责本部门测试工具、测试环境及测试实验室的规划、建设和使用，包括测试技术的研究与测试工具的研究；

⑧ 负责公司各项规章制度在本部门的监督执行。

2.3.2　软件测试人员应具备的素质

在软件开发和软件测试中，软件开发人员和测试人员的素质(包括心理素质)对软件质量的影响都是很大的，例如：开发人员认为他不会犯错，但任何人都可能犯错；开发人员认为这种错误不能算作错误，事实上质量是由用户来评价的；开发人员认为发现错误是对其工作的否定，实际上这对其工作有很好帮助。

测试人员分为两类：测试工具软件开发工程师和软件测试工程师。前者介于软件开发人员和软件测试工程师之间，负责写测试工具代码，并利用测试工具对软件进行测试，或者开发测试工具，为软件测试工程师服务；后者负责理解产品的功能要求，然后对其进行测试，检查软件有没有错误，决定软件是否具有稳定性，并写出相应的测试计划和测试用例。为了方便，我们将软件测试工程师称为软件测试人员。

前面章节我们提到，软件测试是一项复杂而艰巨的任务，软件测试工程师的目标是尽早发现软件缺陷，以便降低修复成本。软件测试员是客户的眼睛，是最早看到并使用软件的人，所以应当站在客户的角度，代表应用客户说话，及时发现问题，力求使软件功能趋

于完善。

很多比较成熟的软件公司都把软件测试视为高级技术职位。软件测试员的工作与程序员的工作对软件开发所起的作用是相当的。虽然软件测试员不一定是优秀的程序员，但是作为出色的软件测试员应当具备丰富的编程知识，掌握软件编程的基础内容，了解软件编程的过程，这无疑对出色完成软件测试任务具有很大的帮助。

对于测试人员，技术能力是相当重要的，但素质培养则更为重要，而这些素质具体地体现在以下几个方面：

1. 端正对软件测试工作的认识

测试人员可能认为软件测试不可能发现所有错误而责任心不够，这需要对他们进行职业教育，实施激励措施；另外，很多测试人员认为测试没有创造性、枯燥，这时就需要引导他们不断总结测试经验，培养发现问题的敏锐度，提升个人价值和权威；还有，测试人员可能认为测试所需要或用到的技术比开发人员差，自信心不足，这就需要消除他们认识上的错误，测试是技术和经验的结合，测试人员能够多快好省地发现错误或缺陷，就能够最大限度地节省时间和费用，从而创造巨大的价值；最后，测试人员认为软件测试就是给人挑毛病，招人厌恶，这就需要对测试人员进行职业教育，让他们意识到软件测试人员的任务就是站在使用者的角度，代表用户通过不断地使用和攻击刚开发出来的软件产品，尽量多地找出产品中存在的问题或错误(bug)，用户满意就是成功。

另外，在测试过程中要做到对事不对人。也就是说，测试人员应该把精力集中在查找错误上面，而不是放在找出是开发小组中哪个成员引入的错误。这样可以保证测试的否定性结果只是针对产品，而不是针对编程人员。也就是说，要使用一种公正和公平的方式指出具体错误，这对于测试工作是有益的。一般来说，武断地对产品进行攻击是错误的。

最后，干测试工作很容易使人变得懒散。只有那些具有自我督促能力的人才能够使自己的工作都能高质量完成。

2. 具有较强的沟通能力、外交能力和移情能力

优秀的测试工程师必须能够和测试涉及的所有人进行沟通，具有与技术和非技术人员进行交流的能力。机智老练和外交手法有助于维护与开发人员的协作关系，幽默感同样也是很有帮助的。测试工程师既要和用户谈得来，又要能和开发人员说得上话，要达到这个目标是很难的，因为这两类人没有共同语言。和用户谈话的重点必须放在系统可以正确地处理什么和不可以处理什么上，尽量不使用专业术语。而和开发者交流时，要尽量使用专业术语，对用户反馈的相同信息，测试人员必须重新组织，以另一种方式表达出来。测试小组的成员必须能够同等地同用户和开发者沟通。特别是与开发人员沟通时，测试人员要善于表达观点，表明软件缺陷为何必须修复，并通过实际演示力陈观点。

当测试人员告诉编程人员他出了错时，就必须使用一些外交方法。机智老练和外交手法有助于维护与开发人员的协作关系，测试者在告诉开发者他的软件有错误时，也同样需要一定的外交手腕。如果采取的方法过于强硬，对于测试者来说，在以后和开发部门的合作方面就相当于赢了战争却输了战役。在遇到狡辩的情况下，幽默的批评将是很有帮助的。

和被测软件系统开发有关的所有人员都处在一种既关心又担心的状态之中。用户担心

将来使用不符合自己要求的系统，开发者则担心由于系统要求不正确而使他不得不重新开发整个系统，管理部门则担心系统突然崩溃而使声誉受损。测试者必须和每一类人打交道，因此需要测试小组的成员对他们每个人都具有足够的理解和同情，具备了这种能力可以将测试人员与相关人员之间的冲突和对抗减少到最低程度。

3. 掌握比较全面的技术

很多情况下，开发人员对那些不懂技术的人持一种轻视的态度。一旦测试小组的某个成员做出了一个比较明显的错误断定，可能会被夸张地到处传扬，那么测试小组的可信度就会受到影响，其他正确的测试结果也会受到质疑。再者，由于软件错误通常依赖于技术，或者至少受构造系统所使用技术的影响，因此测试人员要掌握编程语言、系统构架、操作系统的特性、网络、表示层、数据库的功能和操作等知识，还应该了解系统是怎样构成的，明白被测试软件系统的概念、技术，既要建立测试环境、编写测试脚本，又要会使用软件工程工具。要做到这些，需要有几年以上的编程经验以及对技术和应用领域的深刻理解。

4. 测试中要做到“五心”

专心：主要是指测试人员在执行测试任务的时候要专心，不可一心二用。经验表明，高度集中精神不但能够提高效率，还能发现更多的软件缺陷，业绩最棒的往往是团队中做事精力最集中的那些成员。

细心：主要是指执行测试工作时要细心，认真执行测试，不可以忽略一些细节。某些缺陷如果不细心很难发现，例如一些界面的样式、文字等。

耐心：很多测试工作有时候显得非常枯燥，需要很大的耐心才可以做好。如果比较浮躁，就无法做到专心和细心，这将让很多软件缺陷从你眼前逃过。

责任心：责任心是做好工作必备的素质之一，测试工程师更应该将其发扬光大。如果测试中没有尽到责任，甚至敷衍了事，这将会把测试工作交给用户来完成，很可能引起非常严重的后果。

自信心：自信心是现在多数测试工程师都缺少的一项素质，尤其在面对需要编写测试代码等工作的时候，往往认为自己做不到。要想获得更好的职业发展，测试工程师应该努力学习，建立能解决一切测试问题的信心。

5. 要有很强的记忆力、怀疑精神和洞察力

好的测试者应有能力将以前遇到过的类似错误从记忆深处挖掘出来，这一能力在测试过程中的价值是无法衡量的。因为许多新出现的问题和我们已经发现的问题相差无几。

通常，开发者会尽他们的最大努力将所有的错误解释过去。测试者必须聆听每个人的说明，但必须保持高度警惕、怀疑一切，直到得出自己的分析结果或亲自测试之后，才能做出决定。

好的测试工程师具有测试是为了破坏的观点、捕获用户观点的能力，以及追求质量、关注细节的能力，还应具有应用的高风险区的判断能力，这样就有可能进行针对性测试。

6. 具有探索、创新和挑战精神，努力追求完美

首先，测试人员不要害怕进入陌生环境，要勇于探索。当然，前提是测试人员要

有较强的学习能力，可以用最快的速度进入一个新的行业领域。

另外，测试人员要能够想得出富有创意甚至超常的手段来寻找软件中潜在的各种错误和缺陷。

最后，优秀的测试工程师在开发测试用例时要有敢于挑战的精神，要想方设法找到隐藏在深处的错误和覆盖所有的分支及路径。

通常在测试的过程中，测试人员会碰到转瞬即逝或难以重建的软件缺陷，这时候不要心存侥幸，而是要尽一切可能去寻找，尽力接近目标，力求完美。

7. 测试人员在测试时需要注意的事项

(1) 永远不要许诺或保证什么。在任何时候都不要表露出有了测试人员或者有了像你一样的测试人员，产品绝对没有任何问题。这是在自己打自己的嘴，测试人员要给自己留一条退路，要表露出谦虚的一面，尽量少在用户使用时发现问题，要竭尽全力做好测试工作。

(2) 文档反映了自己的精神面貌。测试人员的任何文档代表的是你本人，所以文档一定要写得漂亮，即要求格式、版面整齐，没有错别字，语言通顺，表达清楚，没有歧义，一般的技术人员都能读懂你的文档。

(3) 要学会逆向思维。开发人员一般都是从正面满足需求，较少去考虑不满足需求的部分，测试人员就要逆向思维考虑，有哪些是开发人员没有考虑到的、不满足需求的部分。

(4) 编写缺陷一定要保证重现。在保证重现缺陷的时候，要注意缺陷不要描述太啰嗦，一般在 3 个步骤内完成操作。

(5) 测试要依据需求，关注对用户不利的缺陷。离开了需求，叫做根本没有真正测试被测项目。另外，在测试中要更多地考虑用户能否正确、完整地使用被测软件，用户使用这套软件能够给他们的工作带来好处。不要过多考虑用户不在意的问题。

(6) 尽量使用测试工具。完全的手工测试过程是非常浪费时间和资源的，所以测试人员应该根据公司的实际情况适当地引入测试工具。一般情况首先引入的是测试管理工具，把整个测试过程管理起来，然后考虑其他测试工具。

(7) 牢记服务意识。测试人员是服务人员，整个项目组的人都是测试人员服务的对象，针对不同的人，我们应该提供不同的帮助与协助。

总之，测试工作对软件测试人员有很高的素质要求，比如：包括坚持原则在内的责任心，包括好奇心在内的怀疑精神和学习能力，包括与用户和项目组人员在内的沟通能力，包括耐心和记忆力在内的专注力，包括经验、逻辑思维能力和敏感度在内的洞察力，以及团队精神等。另外，除了素质要求外，对软件测试人员还有很高的技术要求，比如：计算机操作能力，测试环境搭建能力，一定的编程基础(如果对编程机制、实现架构有一定的了解，会对测试工作很有帮助，发现很多更深层次的问题)，测试基本理论和方法(如掌握测试的基本流程与基本概念，较强的文档能力，会撰写测试报告，会设计、编写测试用例，熟悉测试工具，能够执行测试并跟踪错误等)。

2.3.3　软件测试的就业前景

中商情报网数据显示：2016 年 1 月至 10 月中国软件产业企业个数有 41676 家，其中，软件业务收入为¥39073.04 亿元，同比增长 15.2%；软件产品收入为¥12054.5 亿元，同比增

长 12%；信息系统集成服务收入为¥20343.5 亿元，同比增长 15%。

据前程无忧招聘网统计，目前，国内 120 万软件从业人员中，真正能担当软件测试职位的不超过 5 万人，软件测试人才缺口已超过 20 万并向 30 万大关急速挺进。在中华英才网发布的 2016 年十大热门职业中，软件测试工程师名列前茅。人才的极度匮乏令许多 IT 企业不得不延缓甚至停止项目，为企业发展带来消极影响，但对人才就业却有积极意义，人才供不应求让软件测试人员的就业竞争压力明显小于同类其他职业。

微软公司软件测试工程师对外透露，在微软内部，软件测试工程师和开发工程师的比例基本维持在 1:1，而国内其他软件企业中这一比例仅在 1:5~1:8 之间。招一名高水平的软件测试人员比招博士还难！不少企业发出类似的感叹。但随着中国改革开放的不断深入，不难看出，汽车、电子产品等都有了飞速的发展，软件测试行业更是如此。相信不久的将来，国内软件测试人员与开发人员的比例将会达到甚至超出 1:1。

为了吸引更多的人才，企业纷纷采取高薪策略。据统计，刚入行的软件测试人员，起步月薪大多在 4000~5000 元，高于同龄人 1000~2000 元的薪资水平，另外还可享受带薪年假、内部培训、住房公积金等福利待遇，工作 2 到 3 年后，月薪大约在 8000~13000 元，甚至超出很多相同服务年限的软件开发人员的薪资水平。

与企业高薪难觅良将形成鲜明对比的是，高学历人才难找合适的工作，薪酬持续走低。据中华英才网的调查显示，我国毕业生月平均收入涨幅缓慢。受市场规律影响，人才势必会朝着需求旺盛的职业流动。但流动的过程并非一帆风顺，人才首先面对的就是职业专业性的考验，这一点在软件测试人才身上体现得尤为明显。

软件测试人员最大的优势就是发展方向多。一方面，由于工作的特殊性，软件测试人才更强调经验积累，测试人员不但需要对软件的质量进行检测，而且对于软件项目的立项、管理、售前、售后等领域都要涉及，这样在几年的测试经验背景下，可以逐步转向管理或资深测试工程师，担当测试经理或 QA 部门主管，也可以横向发展为项目经理，所以发展前景广阔，职业寿命更长；另一方面，由于国内软件测试工程师人才奇缺，并且一般只有大中型企业才会单独设立软件测试部门，所以很有保障，待遇普遍较高。正因为如此，软件测试已经成为现在求职者十分关注的职业。

目前，软件测试行业正处在蓬勃发展的初期，未来 5 至 10 年将是发展的高潮期。

习题和思考题

1. 什么是软件测试？软件测试包含哪些概念？我们如何对软件测试的质量进行度量？软件测试与软件开发之间有什么关系？

2. 软件测试工作包括哪些内容？它是怎样的一个流程？测试工具对测试工作有何支持？目前人们对测试工作有哪些不正确的认识？

3. 你怎么看待软件测试？软件测试是一个什么样的行业？如果想从事软件测试工作，怎样做职业准备或规划？

4. 通常企业对软件测试工程师的素质和技能要求有哪些？怎样才能在软件测试职业上获得最佳发展？

5. 列出国内外有关软件测试的网站(用搜索引擎)，并描述它们各自的特点。

6. 给出国内市场对软件测试工程师的需求情况，以及能力要求。

7. 软件测试是一个独立的过程，与开发人员无关，这种说法正确与否，为什么？

第3章　软件测试分类与分级

软件测试是一项复杂的系统工程，对软件测试进行类别与级别的划分时，从不同的角度考虑可以有不同的划分方法。对测试进行分类和分级是为了更好地明确测试过程中的测试任务分类和测试过程中存在的不同级别，清楚各个测试阶段和开发过程期间各阶段的对应关系，了解整个测试究竟要完成哪些工作和哪些任务，尽量对测试工作做周全安排，测试任务合理分配，测试资源最佳调度，做到测试管理的科学化。

3.1　软件测试分类

按照全生命周期的软件测试概念，测试对象应该包括软件设计开发的各个阶段的内容，对于需求和设计阶段的测试以及关于文档的测试在前面章节已论述，这里重点讲述开发阶段的软件测试或程序测试。

对于软件测试，可以从不同的角度进行分类。例如：从是否关心软件内部结构和具体实现的角度划分，软件测试可以分为“白盒”测试、“黑盒”测试和“灰盒”测试；从软件开发过程的角度划分，软件测试可以分为单元测试、集成测试、系统测试、验收测试；从是否执行程序的角度划分，软件测试可以分为静态测试和动态测试；从测试执行时是否需要人工干预的角度划分，软件测试可以分为人工测试和自动化测试；从测试实施组织的角度划分，软件测试可分为开发方测试、用户测试(β 测试)、第三方测试。

我国的 GB/T 15532-2008 计算机软件测试规范给出了基于计算机软件配置项的软件测试分类方法，下面我们就这种分类方法进行全面介绍。

3.1.1　计算机软件配置项

计算机软件配置项缩写为 CSCI(Computer Software Configuration Item)，是为独立的配置管理(技术状态管理)而设计的且能满足最终用户要求的一组软件，简称软件配置项。

1. 软件配置项的概念

在软件开发过程中，产生的所有信息构成软件配置，它们是：代码(源代码和目标代码)、文档(需求文档、技术文档、管理文档等)、报告(包括软件测试过程中所产生的许许多多的工作成果，例如测试计划文档、测试用例以及自动化测试执行脚本和测试缺陷数据等)等，它们都应当被保存起来，以便查阅和修改。这些纳入配置管理范畴的工作成果统称为配置项(Configuration Item，CI)，每个配置项的主要属性有：名称、标识符、文件状态、版本、作者、日期等。

在整个软件生命周期内，软件配置管理控制这些软件配置项的投放和变更，并且记录并报告配置的状态和变更要求，验证配置的完整性、正确性和一致性。然而，尽管这些变

动中的绝大部分是合理的，但是在不同的时机做不同的变动，难易程度和影响的结果差别仍旧很大，为了更有效地控制变更，引入了基线的概念。

2. 基线的概念

基线即软件技术状态基线，指需要受到配置管理控制的某个研制阶段终点处的软件成分的技术状态，已经过正式审核和同意，是下一步软件开发的基础。任何一个软件配置项，一旦形成文档并审议通过，即成为基线。对已成为基线的软件配置项进行修改时，必须按照特殊的、正式的过程进行评价，确认其修改；相反，对于未成为基线的软件配置项，可以进行非正式的修改。它的作用是使各阶段工作的划分更加明确化，使本来连续的工作在这些点上断开，以便于检验和肯定阶段成果。每一个基线都是下一步开发的出发点和参考点。基线确定了元素(配置项)的一个版本，且只确定一个版本。一般情况下，基线一般在指定的里程碑处创建，并与项目中的里程碑保持同步。

3. 软件配置管理

软件配置管理通过协调同一软件项目中不同人员的软件工作产品来帮助我们减轻这些问题。它涉及：识别、定义软件配置项并指定基线；控制软件配置项的修改与发行，记录与报告软件配置项的状态和修改请求；保证软件配置项的完整性、一致性和正确性；履行必要的审批手续，控制软件的存储、处理和交付。若缺乏这种控制，在同一软件项目中，不同人员的工作就会发生冲突，结果会导致如下问题：

(1) 同时修改软件：当两人或两人以上共同开发同一软件时，最后一个人的修改很可能损害前面人员所做的工作。

(2) 共享代码：当一个人修改了共享代码后，通常不是所有的人都能知道。

(3) 公共代码：在一些大型系统中，当修改通用的软件功能时，所有使用这些公共代码的人都必须知道，才不会造成不必要的错误。因此，如果缺乏有效的代码管理，就没有办法保证一一找到，并一一提醒所有的使用者。

在软件开发过程中，开发团队通常已将我们要测试的软件按功能和性能要求合理地分解到各个软件配置项中，并划分了软件配置项关键等级且形成清单。我们可以基于这些文档作为软件测试和质量控制的依据。

3.1.2　基于 CSCI 的软件测试分类

GB/T 15532-2008 计算机软件测试规范给出的测试类别是单元测试、集成测试、配置项测试(也称软件合格性测试或确认测试)、系统测试、验收测试和回归测试。我们可根据软件的规模、类型、完整性级别选择执行测试类别。对这些测试类别的描述结构包含：测试对象和目的、测试的组织和管理、技术要求、测试内容、测试环境、测试方法、测试过程和文档要求。

其中软件配置项测试的对象是软件配置项，其测试目的是检验软件配置项与软件需求规格说明的一致性。对软件配置项的测试，要确保其测试工作的独立性——一般由软件的供方组织，由独立于软件开发的人员实施，由软件开发人员配合。

软件配置项测试的技术依据是软件需求规格说明(含接口需求规格说明)。其测试内容主要依据本教程第 7 章叙述的软件质量模型中包括的功能性、可靠性、可用性、效率、维

护性和可移植性中的 27 个质量特性及子特性的可测试项，根据软件需求的选定来进行。当然也可根据被测对象的实际情况进行测试内容的剪裁。

为保证上述测试类别测试的充分性，有必要进行以下种类的测试：功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余物测试和强度测试。

对于需要支持中文本地化的软件，我们还要进行中文能力测试；对应用软件系统在有条件和有要求的情况下进行应用基准测试。

基于质量特性及质量子特性的配置项测试与上述测试类型的对应关系见表 3-1。可根据这些确定软件测试各个级别的测试内容。

表 3-1　测试内容不同分类的对应关系

质量特性分类	质量子特性分类测试内容	对应关系	传统分类测试内容
功能性	适合性方面		功能测试
	准确性方面		
	互操作性方面		功能多余物测试
	安全保密性方面		
	功能性依从方面		边界测试
可靠性	成熟性方面		
	容错性方面		性能测试
	可恢复性方面		
	可靠性依从方面		接口测试
可用性	可理解性方面		
	易学性方面		安全性测试
	可操作性方面		
	吸引性方面		强度测试
	可用性依从方面		
效率	时间特性方面		可靠性测试
	资源利用方面		
	效率依从方面		恢复性测试
维护性	可分析性方面		
	可修改性方面		人机交互界面测试
	稳定性方面		
	可测试性方面		余量测试
	维护性依从方面		
可移植性	适应性方面		配置测试
	易安装性方面		安装性测试
	共存性方面		
	可替换性方面		兼容性测试
	可移植性依从方面		

1. 功能测试的具体要求

功能测试主要对软件需求规格说明中的功能需求进行测试，找出被测实现与需求不一致的地方，确认一致的地方。在进行功能测试时，要求：

(1) 每一个软件功能必须被一个测试用例或一个被认可的异常所覆盖。

(2) 用基本数据类型和数据值测试。

(3) 用一系列合理的数据类型和数据值运行，测试超负荷、饱和及其他“最坏情况”的结果。

(4) 用假想的数据类型和数据值运行，测试其排斥不规则输入的能力。

(5) 每个功能的合法边界值和非法边界值都必须有测试用例专门测试。

2. 性能测试的具体要求

性能测试主要对软件需求规格说明中定义的性能需求进行测试，说明在一定工作负荷和配置条件下，系统的响应时间及处理速度等特性，找出被测实现与性能需求之间的不一致。在进行性能测试时，要求：

(1) 测试程序在获得定量结果时程序计算的精确性。

(2) 测试程序在有速度要求时完成功能的时间。

(3) 测试程序完成功能所能处理的数据量。

(4) 测试程序各部分的协调性，如高速、低速操作的协调性。

(5) 测试软/硬件中的一些因素是否限制了程序的性能。

(6) 测试程序的负载潜力。

(7) 测试程序运行占用空间。

3. 外部接口和人机交互界面测试的具体要求

外部接口和人机交互界面测试主要对平台各个服务域提供的应用编程接口、应用程序接口、外部环境接口以及人机交互界面的符合性和可用性进行测试。在进行外部接口和人机交互界面测试时，要求：

(1) 测试所有外部接口，检查接口信息的格式及内容。

(2) 测试所有人机交互界面提供的操作和显示界面，并以非常规操作、误操作、快速操作来检查界面的可靠性，以最终用户为背景检验界面显示的清晰性。

(3) 如果有用户手册或操作手册，应对照手册逐条进行操作和观察。

4. 强度测试的具体要求

强度测试必须在预先规定的一个时期内，在软件设计能力的极限状态，进而超出此极限状态，运行软件的所有功能。

5. 余量测试的具体要求

测试程序全部存储量、输入/输出通道及处理时间的余量是否满足需求规格说明中的要求。

6. 可靠性测试的具体要求

可靠性测试是在其有使用代表性的环境中，为进行软件可靠性估计而对其进行的功能测试。在进行可靠性测试时，要求：

(1) 软件可靠性测试必须按照使用的概率分布随机地选择测试实例。

(2) 必须保证输入覆盖，包括输入域覆盖(即覆盖重要的输入变量值，并且所有被测输入值域的概率之和必须大于软件可靠度要求)、各种使用功能的覆盖、相关输入变量可能性组合的覆盖(以确保相关输入变量的相互影响不会导致软件失效)、设计输入空间与实际输入空间之间区域的覆盖(即不合法输入域的覆盖)。

(3) 被测软件的测试环境(包括硬件配置和软件支撑环境)应和预期的实际使用环境尽可能一致。

(4) 对于可能导致软件运行方式改变的一些边界条件(如堆栈溢出)和环境条件(如系统加电、掉电、电磁干扰等)，必须进行针对性测试。

(5) 测试时应记录测试结果、运行时间和判断结果。如果软件失效，还应记录下失效现象和时间。

7. 安全性测试的具体要求

安全性测试主要对平台软件配置项的安全性进行测试，说明安全系统是否存在，是否起到应有的作用，是否达到规定的安全级别等。安全性测试的内容包括系统安全评估和系统侵入测试两个部分：系统安全测试主要涉及标识与鉴别、访问控制、审计、特权管理、可信通路、隐蔽通道等，系统侵入测试主要涉及系统脆弱性分析、系统安全漏洞检测等。在进行安全性测试时，要求：

(1) 必须进行软件安全性分析，并且在软件需求说明中明确每一个危险状态及导致危险的可能原因，在测试中全面检验软件在这些危险状态下的反应。

(2) 对安全性关键的软件部件，必须单独测试，以确认该软件部件满足安全性要求。

(3) 对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案必须进行针对性测试。

(4) 测试应尽可能在符合实际使用的条件下进行。

(5) 除在正常条件下测试外，应在异常条件下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态。例如：

- 必须包含硬件及软件输入故障模式测试。
- 必须包含边界、界外及边界接合部的测试。
- 必须包括“0”、穿越“0”以及从两个方向趋近于“0”的输入值。
- 必须包含在最坏情况配置下的最小和最大输入数据率(以确定系统的固有能力以及对这些环境的反应)。
- 操作员接口测试必须包括在安全性关键操作中的操作员错误(以验证安全系统对这些错误的影响)。
- 应测试双工切换、多机替换的正确性和连续性。
- 应测试防止非法进入系统并保护系统数据完整性的能力。

对于测试中发现的缺陷，必须纠正，以消除所有危险或将其风险降到可接受水平。纠

正后的软件应在同样的条件下重新测试，以确保已消除危险和不会出现其他危险。

8. 恢复性测试的具体要求

对于有恢复或重置(RESET)功能的软件，必须验证恢复或重置功能，对每一类导致恢复或重置的情况进行测试。

9. 边界测试的具体要求

测试程序在输入域(或输出域)、数据结构、状态转换、过程参数、功能界限等边界或端点情况下的运行状态。

10. 功能多余物测试的具体要求

验证程序中未附加的软件需求中没有指明的功能及功能边界的不适当。所有输出都应有意义并在软件需求中指明。

11. 安装性测试的具体要求

安装性测试主要对平台软件配置项的可安装性/可卸载性进行测试。安装性测试通过安装/卸载程序或按照安装/卸载规程进行软件配置项的安装/卸载，发现安装/卸载过程的错误，验证软件配置项的可安装性/可卸载性，包括参数装订、程序从软盘装入计算机等。

12. 中文能力测试的具体要求

中文能力测试主要对平台软件配置项的中文支持能力进行测试，验证软件配置项是否能全面、正确地支持和处理中文。在进行中文能力测试时，要求：

1) 测试中英文转换后的正文长度变化是否对软件有影响

短消息和命令名经过翻译后可能需要更多空间，这时正文可能会超出菜单或对话框，正文字符串在内部存储区中也许会溢出，覆盖其他的代码或数据。

2) 测试软件是否能完全处理中西文字符集

通常一个典型的字符集包含 256 个字符，其编号从 0 到 255。其中编号从 32 至 127 的字符为 ASCII 字符，编号从 0 到 31 的符号为低端 ASCII，编号从 128 到 255 的符号为高端 ASCII。从技术上来说，任何将 1 到 254 的数字与符号联系起来的符号集就是一个代码页。在中文代码页与西文代码页之间，编号从 128 到 255 的符号是存在冲突的，对于中文能力测试来说，必须测试在不同的代码页之间切换时高端 ASCII 字符是否能正确显示。

3) 测试对中文是否存在正文过滤现象

有些程序在一个字段中只能接收一定的字符，也许会将高端 ASCII 字符和各种控制码屏蔽掉，这对英语来说是适当的，但对非英语字符来说可能是错误的。因此，必须在能输入字符的任何地方测试一个程序是怎样接收和显示所有字符的。

4) 测试操作系统语言是否支持中文

测试通配符、文件名定界符、文件名约定在不同语言环境下的作用是否正确。

5) 测试中文排序规则是否正确

字符排序规则在各个国家都各不相同。按内部代码值排序在中文里没有任何意义。测

试程序的排序功能是否符合中文习惯。

6) 测试大小写转换功能对中文是否有影响

中文不存在大小写，测试程序的大小写是否只对西文起作用。

13. 应用基准测试的具体要求

应用基准测试主要对平台软件配置项的综合性能进行测试。应用基准测试通过构造一组能够反映某个领域应用特点的典型应用程序，即面向特定应用领域的应用基准程序，并使之在平台上加以运行，来测试平台软件配置项的综合性能，如软件配置项的适用性、准确性、成熟性、稳定性、时间行为、资源行为、适应性、易学性、易操作性等。

应用基准测试主要用来验证平台软件系统对典型应用的综合支持能力，因此，应用基准程序必须是能够反映应用领域特点的典型甚至标准的应用程序。

在进行测试时需要注意三点：

① 必须有交办方人员参加计算机软件配置项测试。

② 全部预期结果、测试结果及测试数据应存档保留。

③ 建立独立的测试小组，进行计算机软件配置项测试。

上述测试种类共包含 13 类。事实上，在实际测试中并非都要完成上述所有测试种类，也不是对任何软件都要完成上述所有测试。究竟必须执行哪些测试，应根据软件的复杂性、关键等级和当前的测试类别选定。例如在某工程中，对此规定如下：

① 单元测试阶段至少完成功能测试、边界测试。

② 部件集成测试阶段至少完成功能测试、性能测试、余量测试、边界测试和接口测试。

③ 软件配置项测试阶段至少完成功能测试、性能测试、余量测试、边界测试和接口测试。

④ 对于安全关键等级为 A、B 级的软件还必须完成强度测试、可靠性测试、安全测试和功能多余物测试。

3.2　软件测试分级

对软件测试的要求、目的、关注点、被测对象、工作产品及测试人员不同，相应的软件测试级别划分或分级是不同的。目前大家常关注的软件测试的有关分级大致包括软件生命周期测试的分级、错误及其对软件测试通过影响的分级、完整性测试的分级、软件测试用例的分级等。通过对软件测试进行有目的的分级，使我们能够有效地控制软件的复杂性，强化测试的针对性或目的性，提高测试管理的科学性，最终确保软件测试的质量。

本节主要介绍软件生命周期的测试分级和软件测试中的错误分级等内容，其他分级内容在其他章节介绍。

3.2.1　软件生命周期的测试分级

我国国标GB/T 15332-2008计算机软件测试规范、军标GJB 2725A-2001附加文件——《军

用软件测评实验室测评过程和技术能力要求》以及相关行业制定的软件测试规范或标准，均按照软件生命周期对软件测试进行了级别划分。在国标中是以测试类别命名，在军标中是以测试级别命名。大部分涉及单元测试或组件测试、集成测试、配置项测试(也称软件合格性测试或确认测试)、系统测试和验收测试等分级内容。

针对不同的测试级别，我们应该明确：

① 不同的测试对象；

② 每个测试级别的测试目的；

③ 测试用例设计的基础(所参考的软件产品或测试需求)；

④ 发现的典型缺陷和失效；

⑤ 测试工具的需求和支持；

⑥ 不同的测试技术和方法。

测试级别可以根据软件的规模、类型、安全性关键等级进行选择。

回归测试可出现在上述每个测试级别中，并贯穿于整个软件生命周期。

1. 组件测试

针对单个软件单元的测试都可以称为组件测试(根据开发人员和编程语言的不同，软件单元可以是模块、单元、程序或功能)。

1) 组件测试方法

在组件测试过程中，经常会用到桩模块、驱动器、模拟器。这是由于一般被测模块不能形成一个完整的可测试系统，因此在组件测试过程中，需要为测试模块开发驱动器和桩模块。驱动器和桩模块是测试过程中使用的软件，不是软件产品的组成部分，也是需要相关费用的。

组件测试包括功能测试和特定的非功能测试，例如：资源行为测试(内存泄漏)、健壮性测试、基于结构的测试(如分支覆盖)等。组件测试的设计输入主要是单元详细规格说明、软件设计规格说明或数据模型等。

在编写代码之前就开始准备测试和自动化测试用例是组件测试常用的一种方法，称为测试驱动的方法或测试驱动开发。

组件测试的任务包括：模块局部数据结构测试、模块参数边界值测试、模块中所有独立执行路径测试，以及模块的各条错误处理路径测试等。

当程序代码编写完成并通过评审和编译检查后，便可开始组件测试。对于这个测试级别，测试是在与开发紧密合作的情况下进行的，通常由开发人员自己来执行组件测试。组件测试主要采用“白盒”测试方法，通常从程序内部结构出发设计测试用例。

2) 组件测试需要考虑的因素

在组件测试中需要考虑各方面的因素，包括：

(1) 检查单元模块自己的接口相关的参数，是组件测试的基础。

(2) 检查局部数据结构，用来保证临时存储在模块内的数据在程序执行过程中的完整性、正确性。

(3) 在模块中应对每一条独立执行路径进行测试，组件测试的基本任务是保证模块中

的每条路径至少执行一次。

(4) 比较、判断与控制流常常紧密相关。

(5) 好的软件设计应能预见各种出错条件，并预设各种出错处理路径，出错处理路径同样需要认真测试。

2. 集成测试

集成测试也叫组装测试、联合测试，是一种旨在暴露接口以及集成组件/系统间交互时存在缺陷的测试。集成测试是组件测试的逻辑扩展，其关注点是对组件之间的接口进行测试，以及检查与系统其他部分相互作用的测试，如操作系统、文件系统、硬件或系统之间的接口。

1) 集成测试类别划分

根据不同的测试对象规模，可分为组件集成测试和系统集成测试。

组件集成测试对不同的软件组件之间的相互作用进行测试，一般在组件测试之后进行。系统集成测试对不同系统之间的相互作用进行测试，一般在系统测试之后进行。

系统集成测试的范围越大，对缺陷的定位越困难，从而增加了系统的风险。为了降低在软件开发生命周期后期发现严重缺陷而产生的风险，集成程度应该逐步增加。

2) 集成测试感兴趣的应该是两个模块的接口，而不是两个模块本身的功能，“黑盒”测试和“白盒”测试的方法都可以应用在集成测试上

集成测试的对象是已经过组件测试的软件单元，集成测试的依据是软件的概要设计规格说明。

集成测试的主要内容是功能性、可靠性、易用性、效率、可维护性和可移植性。

集成测试采用的测试策略是非渐增式集成测试模式和渐增式集成测试模式，具体包括自底向上集成、自顶向下集成、核心系统先行集成、随意集成。

3. 配置项测试

配置项测试的对象是计算机软件配置项(CSCI，以下简称配置项，它们是能够被独立地进行配置、管理的，并能够满足最终用户功能的一组软件)。配置项测试可根据软件测评任务书、合同或其他等效文件及软件配置项的重要性、安全性关键等级对如下要求进行剪裁，但必须说明理由。配置项测试一般应符合以下技术要求：

(1) 必要时，在高层控制流图中做结构覆盖测试。

(2) 应逐项测试软件需求规格说明规定的配置项的功能、性能等特性。

(3) 配置项的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖。

(4) 测试用例的输入应至少包括有效等价类值、无效等价类值和边界数据值。

(5) 应测试配置项的输入/输出及其格式。

(6) 应测试人机交互界面提供的操作和现实界面，包括用非常规操作、无操作、快捷操作、快速操作测试界面的可靠性。

(7) 应测试运行条件在边界状态和异常状态下，或在人为设定的状态下，配置项的功能和性能。

(8) 应按软件需求规格的要求，测试配置项的安全性和数据的安全保密性。

(9) 应测试配置项的所有外部输入/输出接口(包括和硬件之间的接口)。

(10) 应测试配置项的全部存储、输入/输出通道的吞吐能力和处理时间的余量。

(11) 应按照软件需求规格的要求，对配置项的功能、性能进行强度测试。

(12) 应测试设计中用于提高配置项的安全性和可靠性方案，如结构、算法、容错、冗余、中断处理等。

(13) 对安全性关键的配置项，应对其进行安全性分析，明确每一个危险状态和导致危险的可能原因，并对此进行针对性测试。

(14) 对有恢复或重置功能需求的配置项，应测试其恢复或重置功能和平均恢复时间，并对每一类导致恢复或重置的情况进行测试。

(15) 对不同的实际问题应外加相应的专门测试。

4. 系统测试

系统测试是将已经集成好的软件系统作为计算机系统的一部分，与计算机系统硬件、某些支持软件、数据和人员等系统元素结合起来，在实际运行环境下对计算机系统进行一系列严格有效的测试。

系统测试对测试环境的要求是在集成测试完成后，系统已经完全组合起来后进行，应该在尽可能和目标运行环境一致的情况下进行。

系统测试的目的是确认整个系统是否满足系统需求规格说明中的功能和非功能需求，以及满足程度。

常见系统测试包括压力测试、容量测试、性能测试、安全测试、容错测试等。

5. 验收测试

验收测试通常由使用系统的用户进行测试，目的是确保系统功能、系统的某部分或特定的系统非功能特征满足验收准则，发现缺陷不是验收测试的主要目标。

验收测试的主要测试类型有根据合同的验收测试、用户验收测试、运行(验收)测试、现场测试。

6. 维护测试

维护测试是指软件被市场接受后，在运行一段时间后，需要做某些修正、改变或扩展的情况下进行的维护测试。维护测试是在运行的系统上进行的，属于回归测试类型。

3.2.2　软件测试中的错误分级及其应用

软件测试的目的是暴露错误，评价程序的可靠性。而对软件错误进行级别定义或分级，目的就是科学地指导软件测试工作，提高软件测试的目的性，确保软件测试的质量。

1. 错误分级

软件错误分级涉及两个方面：错误分类及错误分级。不同的行业和企业、不同的应用领域以及不同的错误类型，错误的分级方法是不一样的。

1) 错误分类

软件错误分类有很多种方法，具体方法如下：

① 按软件生命周期分类，有用户需求错误、产品需求错误、设计错误、编码错误、数据错误、发行错误；

② 按软件使用分类，有功能错误、性能错误、界面错误、流程错误、数据错误、提示错误、常识错误以及其他错误；

③ 按 GB/T 15532-2008 分类，有程序问题、文档问题、设计问题及其他问题。

2) 错误级别的划分

软件错误级别的划分同样有很多方法，GB/T 15532-2008 将对软件进行全面测试时所发现的错误分为如下级别：

第 1 级错误是有下列行为之一的软件问题：

① 妨碍由基线要求所规定的运行或任务的主要功能的完成；

② 妨碍操作员完成运行或任务的主要功能；

③ 危及人员安全。

第 2 级错误是有下列行为之一的软件问题：

① 给由基线要求所规定的运行或任务的主要功能的完成造成不利影响，以至于降低效能，且没有变通的解决办法；

② 给操作员完成由基线要求所规定的运行或任务的主要功能造成不利影响，以至于降低效能，且没有变通的解决办法。

第 3 级错误是有下列行为之一的软件问题：

① 给由基线要求所规定的运行或任务的主要功能的完成造成不利影响，以至于降低效能，但已知有变通的解决办法。

② 给操作员完成由基线要求所规定的运行或任务的主要功能造成不利影响，以至于降低效能，但已知有变通的解决办法。

第 4 级错误：这种软件问题给操作员带来不方便或麻烦，但不影响所要求的运行或任务的主要功能。

第 5 级错误：所有的其他错误。

2. 错误分组举例

下面是 GB/T 15532-2008 在某企业中软件错误分级的实例化。

第 1 级：严重缺陷。即应用系统崩溃或系统资源使用严重不足：

① 系统停机(含软件、硬件)或非法退出，且无法通过重启恢复；

② 系统死循环；

③ 数据库发生死锁或程序原因导致数据库断连；

④ 系统关键性能不达标；

⑤ 数据通信错误或接口不通；

⑥ 错误操作导致程序中断。

第 2 级：较严重缺陷。即系统因为软件严重缺陷导致下列问题：

① 重要交易无法正常使用、功能不符合用户需求；

② 重要计算错误；
③ 业务流程错误或不完整；
④ 使用某交易导致业务数据紊乱或丢失；
⑤ 业务数据保存不完整或无法保存到数据库中；
⑥ 周边接口出现故障(需要考虑接口时效/数量等综合情况)；
⑦ 服务程序频繁需要重启(每天两次或以上)；
⑧ 批处理报错中断导致业务无法正常开展；
⑨ 前端未合理控制并发或连续单击动作，导致后台服务无法及时响应；
⑩ 在产品声明支持的不同平台下，出现部分重要交易无法使用或错误。
第 3 级：一般性缺陷。即系统因为软件一般缺陷导致下列问题：
① 部分交易使用存在问题，不影响业务继续开展，但造成使用障碍；
② 初始化未满足客户要求或初始化错误；
③ 功能点能实现，但结果错误；
④ 数据长度不一致，无数据有效性检查或检查不合理，数据来源不正确；
⑤ 显示/打印的内容或格式错误；
⑥ 删除操作不给提示；
⑦ 个别交易系统反应时间超出正常合理时间范围；
⑧ 日志记录信息不正确或应记录而未记录；
⑨ 在产品声明支持的不同平台下，出现部分一般交易无法使用或错误。
第 4 级：较小缺陷。即系统因为软件操作不便导致的缺陷：
① 系统某些查询、打印等实时性要求不高的辅助功能无法正常使用；
② 界面错误；
③ 菜单布局错误或不合理；
④ 焦点控制不合理或不全面；
⑤ 光标、滚动条定位错误；
⑥ 辅助说明描述不准确或不清楚；
⑦ 提示窗口描述不准确或不清楚；
⑧ 日志信息不够完整或不清晰，影响问题诊断或分析。
第 5 级：其他缺陷。即系统辅助功能缺陷：
① 缺少产品使用、帮助文档、系统安装或配置方面的信息；
② 联机帮助、脱机手册与实际系统不匹配；
③ 系统版本说明不正确；
④ 长时间操作未给用户进度提示；
⑤ 提示说明未采用行业规范语言；
⑥ 显示格式不规范；
⑦ 界面不整齐；
⑧ 软件界面、菜单位置、工具条位置以及相应提示不美观，但不影响使用。

习题和思考题

1. 什么是软件配置项？什么是软件配置项测试？软件配置项测试有哪些测试内容？如何对它们进行分类并进行测试种类的选择？

2. 对单元测试、集成测试、配置项测试(也称软件合格性测试或确认测试)、系统测试、验收测试和回归测试进行名词解释。

3. 对功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余物测试和强度测试进行名词解释。

4. 中文能力测试主要测试哪些内容？

5. 什么是软件测试分级？简述软件生命周期测试分级及软件测试错误分级的思想及其应用。

第 II 部分　软件测试过程篇

统计数据表明，大多数软件开发项目的失败，并不是软件开发技术方面的原因，而是由于不适当的管理造成的。同样，软件测试项目能否取得成功或者能否高质量地完成，其重要因素就是软件测试管理。本部分涉及两方面的管理内容：软件缺陷管理和软件测试过程管理，它们是软件测试管理的重要基础，也是软件测试过程中必不可少的最重要组成部分。

第4章　软件缺陷管理

软件危机最重要的特征就是：软件产品的质量靠不住，错误一大堆，难以维护。特别是大型软件，其错误更多，维护更加困难。因此，为了保证软件正常运行，必须对软件中存在的缺陷进行检查或测试，对发现的错误进行有效的管理，从而为软件缺陷或错误的消除，或为软件质量的评价及软件开发的决策提供依据。

4.1　软件缺陷

1991 年海湾战争中，美军使用爱国者导弹拦截伊拉克飞毛腿导弹，出现过几次拦截失败事件，后经查明是软件计时系统的累积误差所致，该软件缺陷是一个很小的系统时钟错误积累起来造成十几个小时的延迟，致使跟踪系统准确度丧失，最终导致严重的后果。事实上，软件系统的可靠性是很难保证的，几乎没有不存在错误或缺陷的软件系统。这是因为，软件错误或软件缺陷是软件产品的固有成分，是软件“与生俱来”的特征。不管是小程序还是大型软件系统，无一例外地都存在缺陷，这些软件缺陷，有的容易表现出来，有的隐藏很深难以发现，有的对使用影响轻微，有的会造成财产甚至生命的巨大损失。

4.1.1　软件缺陷的定义

1. 什么是软件缺陷

在需求分析和设计过程中，需求规格说明在某种程度上与用户要求不符，或者设计中存在一些错误；在写完程序进行编译时会出现语法错误、拼写错误或程序语句错误；软件完成后，应有的功能不能使用；或者软件在交付使用后，出现一些在测试时没有发现的问题，造成软件故障；等等。所有这些都可以称为软件缺陷，可以用图 4-1 表示。

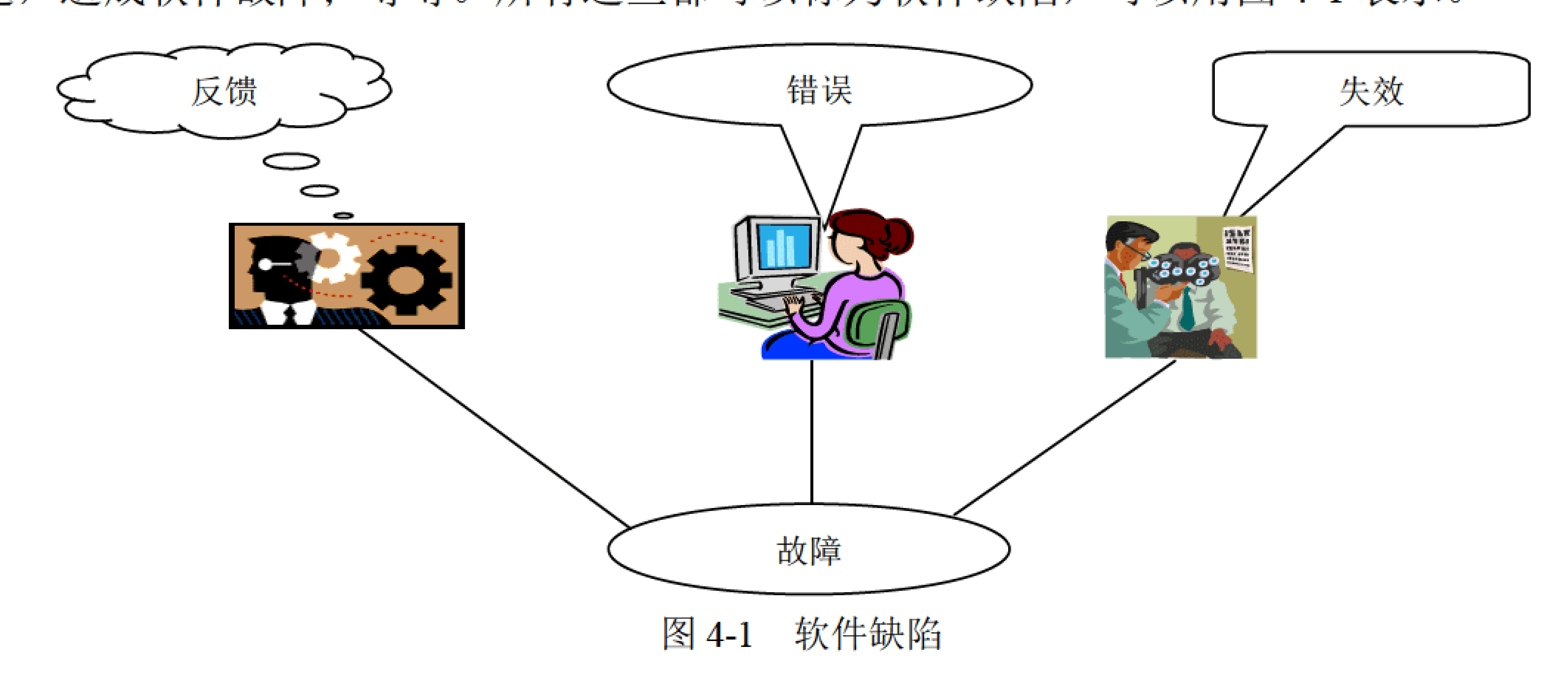


图 4-1　软件缺陷

软件缺陷包括检测缺陷和残留缺陷。检测缺陷是指软件在用户使用之前被检测出的缺陷；残留缺陷是指软件发布后存在的缺陷，包括在用户安装前未被检测出的缺陷以及检测出但未被修复的缺陷。用户使用软件时，因残留缺陷引起的软件失效症状称为软件故障。软件故障是指软件没有表现出人们所期待的正确的结果。软件失效是指软件出现的一些状态，例如：①功能部件执行其功能的能力的丧失；②系统或系统部件丧失了在规定的限度内执行所要求功能的能力；③程序操作背离了程序需求。缺陷是这些故障和失效的源头，要想获得高质量的软件，就要从消除软件缺陷着手，进行缺陷的预防、检测和消除工作。

软件缺陷简单地说就是存在于软件(文档、数据、程序)之中的那些不希望或不可接受的偏差，即软件质量问题。按照一般的定义，只要符合下面 5 条规则中的一条，就叫做软件缺陷：①软件未实现产品说明书要求的功能；②软件出现了产品说明书指明不应该出现的错误；③软件实现了产品说明书未提到的功能；④软件未实现产品说明书虽未明确提及但应该实现的目标；⑤软件难以理解、不易使用、运行缓慢或者——从测试员的角度看——最终用户会认为不好。

总之，软件缺陷(defect 或 bug)是软件开发过程中的“副产品”，缺陷会导致软件产品在某种程度上不能满足用户的需要，导致对软件产品预期属性的偏离，造成用户使用的不便。SW-CMM 将其定义为：“系统或系统成分中能造成它们无法实现其被要求的功能的缺点。如果在执行过程中遇到缺陷，可能导致系统的失效。”

2. 软件缺陷带来的风险

软件缺陷会为系统带来一系列的风险，试想：如果软件的某些代码产生了错误，会导致什么样的结果？未被验证的数据交换如果被接受，又会带来怎样的结果？如果文件的完整性被破坏，那么是否还能保证系统符合用户的要求？软件是否还能如期按要求交付使用？当系统发生严重故障时，如果不能保证系统被安全恢复(恢复成被备份时的状态)，那么系统将会完全崩溃；因某种需求，要暂停系统的运行，而系统又没有办法停止，带来的后果可想而知；进行维护工作时，系统性能下降到不能接受的水平，从而使得用户的需求得不到满足，甚至因此带来严重的影响；系统的安全性是否有保证，尤其是对于一些特殊的系统，若安全性得不到保障，这个软件可以说是不合格的；系统的操作流程是否符合用户的组织策略和长远规划；若系统的运行不可靠、不稳定，相信用户也不会满意；系统是否易于使用；系统是否便于维护；每一个系统都不是封闭、完全独立的，若系统不能与其他系统相连或者很难与其他系统相连，都是不合格的产品。这其中的任何一点都会造成软件开发的失败，例如软件不能正常使用，引起灾难性事件，等等。

3. 软件缺陷产生的原因

现在我们知道了什么是软件缺陷，也了解了软件缺陷的影响，但是软件缺陷为什么会出现呢？事实上导致软件产生缺陷有 9 类原因：①不完善的需求定义；②客户与开发者通信失败；③对软件需求的故意偏离；④逻辑设计错误；⑤编码错误；⑥不符合文档编制与编码规定；⑦测试过程不足；⑧规程错误；⑨文档编制错误。

针对上述九类原因，经过长时间的调查研究，得出了一个令人惊讶的结论：大多数软件缺陷并不是由于编码造成的，导致大多数软件缺陷产生的最大原因在需求分析阶段，其

次是在软件设计阶段，如图 4-2 所示。

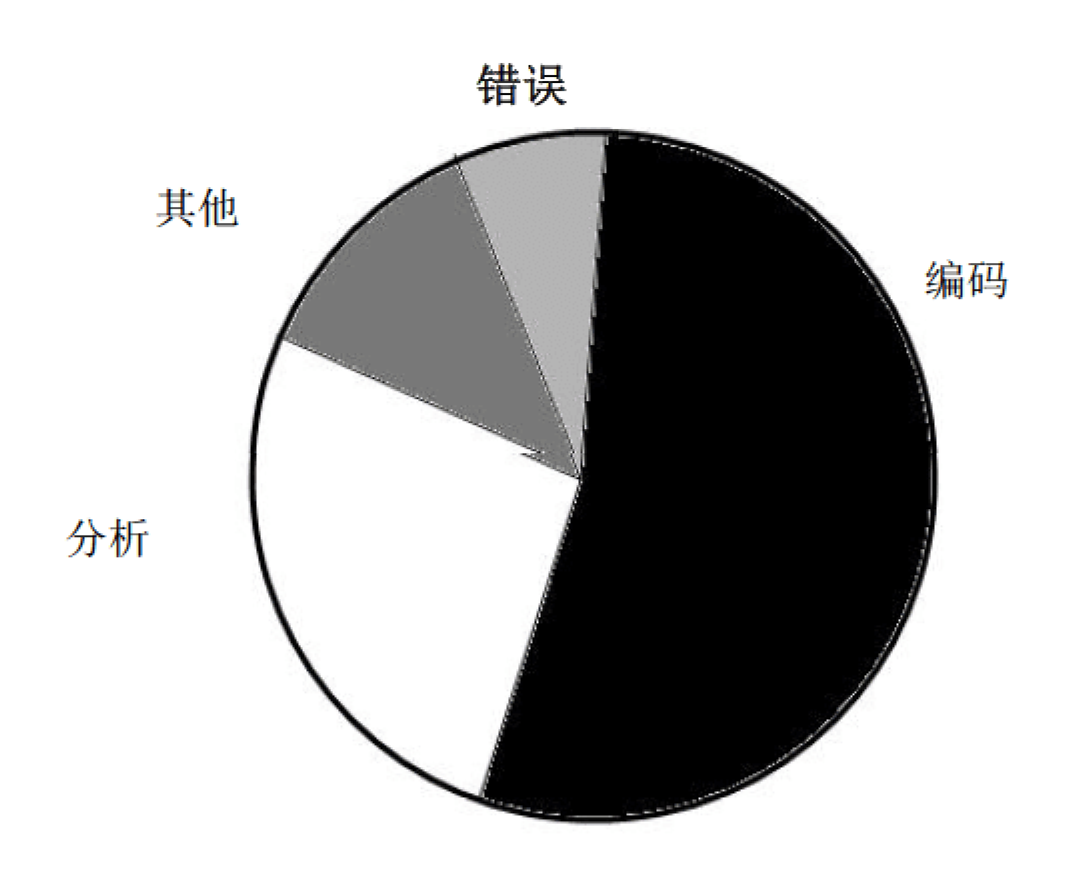


图 4-2　软件缺陷产生的原因很多，其中主要原因是需求分析

1) 需求分析导致缺陷产生的原因

需求分析(也可以说成需求规格说明)成为造成软件缺陷出现的最大来源是有原因的。软件需求规格说明书描述了系统应该具有哪些功能，不应该具有哪些功能，功能的操作性如何，性能如何等具体规格。它是开发初期最重要的过程文档，也是后期开发与测试的重要依据，可以说是开发流程与测试流程的输入。根据过程理论，“正确的输入，正确的过程，正确的解决方案，将会产生正确的结果”，如果一开始输入就不正确，那么经过过程的处理后，缺陷/错误会被放大，同时修复成本会显著上升，人力、物力、时间将会被大量耗费。所以从早期就开始对需求规格说明书进行审查并基线化是必需的，同时测试人员在需求基线化之前应该被安排到流程中，参与评审，尽早从客户/测试的角度找出所有不合理/不明确/不可行的需求，减少后期的开发与测试成本。测试人员以及质量人员在开发初期是比较重要的角色，责任比较重大，应当负起责任。由此可见，需求规格说明书是非常重要的文档。

另外，在软件开发之初，由于客户和开发者之间的通信失败，造成需求规格说明的不完善或是对软件需求的偏离；而后在开发过程中因需求规格说明的不全面或经常变更，再加上整个开发小组不能很好地沟通，造成设计和编码与需求规格说明之间的不一致等。

2) 设计导致缺陷产生的原因

设计是缺陷产生的另一个主要来源，设计是软件开发人员规划软件的过程，就好比建筑师在建筑物建造之前要绘制建筑蓝图，在这个过程中可能会存在一些逻辑错误。

另外设计也会产生变化，会修改，再加上整个开发小组不能很好地沟通，所有这些就造成软件缺陷的产生。

3) 编码导致缺陷产生的原因

软件缺陷在编码阶段出现，通常是因代码错误而造成，由于软件复杂、文档不足、进度压力、普通的低级错误或是因程序员的思维定式而引起。在编码完成后，软件出现错误，经常听到程序员说的话就是：“这是按要求做的，若是有人早告诉我，我肯定就不会这样编写了”。因此，许多软件缺陷似乎是在软件编码阶段出现的，但实际上是由需求规格说明或软件设计造成的。

剩下的原因可以归为一类。可能由测试错误引起，也可能是因为对需求理解错误等，但是这部分只占极小的比例。

4. 为什么软件缺陷难以测试或发现

你知道了什么是软件缺陷、软件缺陷的风险，明白了软件缺陷产生的原因。由此也可以明白软件测试人员的任务就是：尽可能早地找出软件缺陷、并确保其得以修复。现在软件测试已经在软件开发过程中占很大的比重，软件开发人员也越来越重视软件测试，但是为什么仍旧存在那么多的软件缺陷不能被测试到呢？

由于软件是由人来完成的，以目前的技术无法避免错误，有错是软件的属性，这是很难改变的。现在人们已经逐步认识到，所谓的软件危机实际上仅是一种状况，那就是软件中有错误/缺陷，正是这些错误/缺陷导致软件开发在成本、进度和质量上的失控。因此，必须面对现实，避免软件中错误/缺陷的产生和消除已经产生的错误/缺陷，使程序中的错误/缺陷密度达到尽可能低的程度。

但是，由于软件固有的特性，使得我们很难找出或排除软件中的错误/缺陷，因为：①软件错误/缺陷很难看到；②软件错误/缺陷看到了但很难抓到；③软件错误/缺陷抓到了但无法修改或很难修改；④人们无时无刻都可能犯错误，使得软件中存在错误/缺陷。

典型的软件错误/缺陷类型有需求定义错误、需求解释错误、设计记录错误、设计说明错误、编码说明错误、程序代码编写错误、数据输入错误、测试错误、问题修改错误、正确的结果由其他的缺陷产生。

因此，尽管软件测试的目的是尽可能多地发现软件中潜在的缺陷，但并不能保证所有的缺陷都被发现。有些缺陷在测试过程中是无法被发现的。

另外，有些软件缺陷虽然在测试过程中可以被发现，但是测试人员不能使其重现，也就是看得见、抓不到，从而使得缺陷不能得到修复。

最后，尽管原则上软件缺陷在任何时候都必须得到修复，但由于没有足够的时间、不算真正的软件缺陷、修复的风险太大等原因，产品开发小组可以决定对一些软件缺陷不作修复。

4.1.2　软件缺陷描述

从软件缺陷的定义中，我们可以知道判断缺陷的唯一标准就是看其是否符合用户的需求。在大型软件开发过程中，会出现成千上万甚至更多软件缺陷，要确定这些缺陷怎样描述、分类和跟踪或监控，以确保每个被发现的缺陷都能够及时得到处理。例如：确保每个被发现的缺陷都能够被解决；收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段；收集缺陷数据并进行数据分析，作为组织的过程财富。

在运行良好的组织中，缺陷数据的收集和分析是很重要的，从缺陷数据中可以得到很多与软件质量相关的数据。

对缺陷进行跟踪、监控或管理的基本流程是：首先要准确地描述缺陷，然后对各种缺陷进行分类。在此过程中，通过对缺陷进行分类，可以迅速找出哪一类缺陷的问题最大，怎样集中精力预防和排除这一类缺陷，并在这几类缺陷得到控制的基础上，再进一步找到新的容易引起问题的其他几类缺陷。

对软件缺陷进行有效描述涉及如下内容：

1. 可追踪信息

缺陷 ID(唯一的缺陷 ID，可以根据该 ID 追踪缺陷)。

2. 缺陷的基本信息

缺陷的基本信息有如下几部分内容：

(1) 缺陷标题：描述缺陷的标题。

(2) 缺陷的严重程度：描述缺陷的严重程度。一般分为“致命”、“严重”、“一般”、“建议”四种。

(3) 缺陷的紧急程度：描述缺陷的紧急程度。等级从 1 到 4，1 是优先级最高的等级，4 是优先级最低的等级。

(4) 缺陷提交人：缺陷提交人的名字(邮件地址)。

(5) 缺陷提交的时间：缺陷提交的时间。

(6) 缺陷所属项目/模块：缺陷所属的项目和模块，最好能较精确地定位至模块。

(7) 缺陷指定的解决人：缺陷指定的解决人，在缺陷“提交”状态为空，在缺陷“分发”状态下由项目经理指定相关开发人员修改。

(8) 缺陷指定的解决时间：项目经理指定的开发人员修改此缺陷的截止时间。

(9) 缺陷处理人：最终处理缺陷的处理人。

(10) 缺陷处理结果描述：对处理结果的描述，如果对代码进行了修改，要求在此处体现出修改。

(11) 缺陷处理时间：对缺陷进行处理的时间。

(12) 缺陷验证人：对被处理缺陷验证的验证人。

(13) 缺陷验证结果描述：对验证结果的描述(包括通过或不通过的结论)。

(14) 缺陷验证时间：对缺陷进行验证的时间。

3. 缺陷的详细描述

对缺陷描述的详细程度直接影响开发人员对缺陷的修改，描述应该尽可能详细。

4. 测试环境说明

对测试环境的描述。

5. 必要的附件

对于某些文字很难表达清楚的缺陷，使用图片等附件是必要的。

6. 从统计的角度出发

从统计的角度出发，还可以添加“缺陷引入阶段”、“缺陷修正工作量”等条目。

对软件缺陷的描述是后面要论述的软件缺陷报告的基础部分，也是测试人员就一个软件问题与开发小组交流的最初且最好的机会。好的描述需要使用简单的、准确的、专业的语言来抓住缺陷的本质。否则，就会使信息含糊不清，可能会误导开发人员。

清晰准确的软件缺陷描述可以减少软件缺陷从开发人员返回的数量，提高软件缺陷修

复的速度，使每一个小组能够有效地工作，提高测试人员的信任度；同时，也可以得到开发人员对清晰的软件缺陷描述的有效响应，加强开发人员、测试人员和管理人员的协同工作。

4.1.3　软件缺陷的分类

对软件缺陷进行分类，分析产生各类缺陷的软件过程原因，总结在软件开发过程中不同软件缺陷出现的频度，制定对应的软件过程管理与技术两方面的改进措施，是提高软件组织的生产能力和软件质量的重要手段。

1. 软件缺陷分类方法

对缺陷分类是在缺陷描述的基础上进行的。在对缺陷进行分类之前，我们首先要定义缺陷的属性，即属性名称描述、缺陷标识(标记某个缺陷的一组符号，每个缺陷必须有一个唯一的标识)、缺陷类型(根据缺陷的自然属性划分的缺陷种类)、缺陷严重程度(因缺陷引起的故障对软件产品的影响程度)、缺陷优先级(缺陷必须被修复的紧急程度)、缺陷状态(缺陷的跟踪修复过程的进展情况)、缺陷起源(缺陷引起的故障或事件第一次被检测到的阶段)、缺陷来源(引起缺陷的起因)、缺陷根源(发生错误的根本因素)。

软件缺陷的分类方法繁多。各种分类方法的目的不同，观察问题的角度和复杂程度也不一样。几种有代表性的软件分类方法如下：

1) Putnam 分类法

Putnam 等人提出的分类方法将软件缺陷分为六类：需求缺陷、设计缺陷、文档缺陷、算法缺陷、界面缺陷和性能缺陷。

2) 军标分类法

我国国家军用标准 GJB 437 根据军用软件错误的来源将软件错误分为三类：①程序错误，运行程序与相应的文档不一致，而文档是正确的；②文档错误，运行程序与相应的文档不一致，而程序是正确的；③设计错误，虽然运行程序与相应的文档一致，但是存在设计缺陷，可能产生错误。

该分类方法可以分析软件缺陷的来源和出处，指明修复缺陷的努力方向，为软件开发过程各项活动的改进提供线索。分类简单是该分类方法的显著特点。因为分类方法简单，所以提供的缺陷相关信息对具体的缺陷修复工作贡献有限。

3) Thayer 分类法

Thayer 软件错误分类法是按错误性质分类，它利用测试人员在软件测试过程中填写的问题报告和用户使用软件过程中反馈的问题报告作为错误分类的信息。它包括 16 类，在这 16 类之下，再分为 164 个子类。16 类有计算错误、逻辑错误、I/O 错误、数据加工错误、操作系统和支持软件错误、配置错误、接口错误、用户需求改变(指用户在使用软件后提出软件无法满足的新要求产生的错误)、预置数据库错误、全局变量错误、重复错误、文档错误、需求实现错误(指软件偏离了需求说明产生的错误)、不明性质错误、人员操作错误、问题(指软件问题报告中提出的需要答复的问题)。

该分类方法特别适合指导开发人员的缺陷消除和软件改进工作。通过对错误进行分类统计，可以了解错误分布状况，对错误集中的位置重点加以改进。该方法分类详细，适用

面广，当然分类也较为复杂。该分类方法没有考虑造成缺陷的过程原因，不适用于软件过程改进活动。

4) IEEE 分类法

电气和电子工程师学会制定的软件异常分类标准(IEEE Standard Classification for Anomalies 1044-1993)对软件异常进行了全面的分类。该标准描述了软件生命周期各个阶段发现的软件异常的处理过程。分类过程由识别、调查、行动计划和实施处理四个步骤组成，每一步骤包括三项活动：记录、分类和确定影响。异常的描述数据称为支持数据项。分类编码由两个字母和三个数字组成。如果需要进一步分类，可以添加小数。例如 RR 324、IV 321.1。RR 表示识别步骤，IV 表示调查步骤，AC 表示行动计划步骤，IM 表示确定影响活动，DP 表示实施处理步骤。分类过程的四个步骤都需要支持数据项。由于每一项都有各自的支持数据项，该标准不强制规定支持数据项，但提供了各个步骤相关的建议支持数据项。强制分类建立通用的定义术语和概念，便于项目之间、商业环境之间、人员之间的交流沟通。可选分类提供对于特殊情况有用的额外细节。在调查步骤，对实际原因、来源和类型进行了强制分类。其中调查步骤将异常类型分为逻辑问题、计算问题、接口/时序问题、数据处理问题、数据问题、文档问题、文档质量问题和强化问题共八大类，下面又分为数量不等的小类。分类细致深入，准确说明了异常的类型。

该分类方法提供了一种统一的方法来对软件和文档中发现的异常进行详细分类，并提供异常的相关数据项来帮助异常的识别和异常的跟踪活动。IEEE 软件异常分类标准具有较高的权威性，可针对实际的软件项目进行裁剪，灵活度高，应用面广。不足之处是没有考虑软件工程的过程缺陷，并且分类过程复杂。但是该方法提供了丰富的缺陷信息。缺陷原因分析活动可以充分利用这些信息进行原因分析。

5) 正交缺陷分类法

正交缺陷分类(Orthogonal Defects Classification，ODC)是 IBM 公司提出的缺陷分类方法。该分类方法提供一个从缺陷中提取关键信息的测量范例，用于评价软件开发过程，提出正确的过程改进方案。该分类方法用多个属性来描述缺陷特征。在 IBM ODC 最新版本里，缺陷特征包括八个属性：发现缺陷的活动、缺陷影响、缺陷引发事件、缺陷载体(Target)、缺陷年龄、缺陷来源、缺陷类型和缺陷限定词。ODC 对这八个属性分别进行了分类。其中缺陷类型被分为八大类：赋值、检验(Checking)、算法、时序、接口、功能、文档、关联(Relationship)。由此看来，它的缺陷类型很简单，开发人员一般根据缺陷类型来修复程序，缺陷类型对于开发人员来说较容易理解，不会引起歧义。

该分类方法分类细致，适用于缺陷的定位、排除、缺陷原因分析和缺陷预防活动。缺陷特征提供的丰富信息为缺陷的消除、预防和软件过程的改进创造了条件。ODC的缺点在于分类复杂，难以把握分类标准，缺陷分析人员的主观意见会影响属性的确定。

正交缺陷分类既是缺陷分类方法，同时又是软件缺陷度量分析方法，介于统计缺陷模型和因果分析方法之间。在成本方面，ODC和定量方法一样较低，在效果上却达到了定性分析的力度。ODC方法的缺陷数据为详细的过程分析奠定了基础，可以完整地分析全部缺陷的现象，做到对缺陷本质特性的分析和预防。

正由于正交缺陷分类的优异之处，ODC自提出后，得到了广泛的发展与应用，全球多

个软件组织都已经接受并使用ODC。近几年，业界也开始研究并使用ODC。作为定量的测量和分析方法，ODC已经成为CMM4/5 的支撑工具之一，为CMM4/5 定量过程管理、缺陷预防等提供有力支持。

2. 软件缺陷分类方法的应用

按照 CMM 和 GJB 5000-2003(CMM)的要求，参照正交缺陷分类法(ODC)，可在软件生命周期的各个阶段，根据相应阶段的评审和测试活动所提交的问题报告(包括各阶段中的问题建议报告)，确认缺陷的发现阶段和注入阶段，对缺陷进行分类。具体分类过程如下：根据评审人员或测试人员提交的缺陷报告(包括问题建议报告)重现缺陷，确认缺陷存在，并确认缺陷相关数据，一般包括发现缺陷的检测活动、引发缺陷的事件、缺陷来源、缺陷症状、缺陷的影响、缺陷类型、缺陷的严重性等，然后根据缺陷数据对缺陷进行分类，将缺陷划归某个产生该缺陷的软件过程。因此，实际应用中的缺陷分类通常是按照缺陷的表现形式、缺陷的严重程度、缺陷的优先级、缺陷的起源和来源、缺陷的根源以及缺陷的生命周期等进行的。

1) 软件缺陷类型标准(即缺陷的表现形式)

(1) 10 F-Function(功能)。影响了重要的特性、用户界面、产品接口、硬件结构接口和全局数据结构；并且设计文档需要正式的变更，如逻辑、指针、循环、递归、功能等缺陷。

(2) 20 A-Assignment(赋值)。需要修改少量代码，如初始化或控制块，如声明、重复命名、范围、限定等。

(3) 30 I-Interface and Timing/Serialization(接口/时序)。与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表相互影响的缺陷。

(4) 40 C-Checking(检查)。提示的错误信息、不适当的数据验证等缺陷。

(5) 50 B-Build/Package/Merge(联编打包)。因配置库、变更管理或版本控制引起的错误。

(6) 60 D-Documentation(文档)。影响发布和维护，包括注释。

(7) 70 G-Algorithm(语法)。算法错误，如语法、标点符号、书写错误等。

(8) 80 U-User Interface(用户界面)。人机交互特性：屏幕格式、确认用户输入、功能有效性、页面排版等方面的缺陷。

(9) 90 P-Performance(性能)。不满足系统可测量的属性值，如执行时间、事务处理速率等。

(10) 100 N-Norms(标准)。不符合各种标准的要求，如编码标准、设计符号等。

2) 按缺陷的严重程度划分

按缺陷的严重程度划分，是指按软件的缺陷对软件质量的影响程度，即缺陷的存在对软件的功能和性能产生怎样的影响，按照严重程度由高到低的顺序可以分为 5 个等级。

(1) Critical：不能执行正常的工作功能或重要功能，或者危及人身安全。

(2) Major：严重地影响系统要求或基本功能的实现，且没有办法更正(重新安装或重新启动该软件不属于更正办法)。

(3) Minor：严重地影响系统要求或基本功能的实现，但存在合理的更正办法(重新安装或重新启动该软件不属于更正办法)。

(4) Cosmetic：使操作者不方便或遇到麻烦，但不影响执行工作或重要功能。

(5) Other：其他错误。

需要说明的是，在具体的软件项目中，要根据实际情况来划分等级，不一定是 5 个等级。如果缺陷数目较少，可以适当地减少等级，而一般的缺陷管理工具会自动地根据具体项目给出默认的缺陷严重程度。

同行评审错误的严重程度被划分为 Major(主要的、较大的缺陷)和 Minor(次要的、小的缺陷)。

3) 按优先级划分

优先级是处理和修正软件缺陷的先后顺序的指标，即哪些缺陷需要优先修正，哪些缺陷可以稍后修正，按照优先级由高到低可以分为 3 个等级：高(high)、中(middle)、低(low)。其中高优先级的缺陷是应该被立即解决的；中优先级的缺陷是指缺陷需要正常排队等待修复或列入软件发布清单；低优先级的缺陷是指缺陷可以在方便的时候被纠正。和缺陷的严重程度一样，优先级的划分也不是绝对的，可以根据具体的情况灵活划分。

例如，在项目开发期间原来标记为中的缺陷随着时间即将用尽，以及软件发布日期临近，可能变为低优先级。作为发现该软件缺陷的测试人员，需要继续监视缺陷的状态，确保自己能够同意对其所做的变动，并进一步提供测试数据或说服别人修正缺陷。

在这里需要说明的是，软件缺陷的严重程度和优先级是含义不同但又相互联系的两个概念，它们从不同的侧面描述了软件的缺陷对软件质量和最终用户的满意度的影响程度和处理方式。

一般说来，严重程度高的缺陷通常具有较高的优先级。因为严重程度高的缺陷对软件质量的影响大，应该优先处理，而严重程度低的缺陷可能只是软件不太完美，可以稍后再做处理。但是严重程度高的缺陷优先级也一定高吗？即缺陷的严重程度和缺陷的优先级一定成正比吗？答案是：不一定！例如：

(1) 严重程度高的优先级不一定高。软件本身是脆弱的，难以理清头绪，犹如一团乱麻。如果修复一个软件缺陷，需要重新修改软件的整体架构，由此可能造成牵一发而动全身——产生其他的缺陷，而且又有紧迫的产品发布等进度压力，修正软件将冒很大的风险，此时即使缺陷的严重程度很高，是否要修正，也仍需要做全面考虑。

(2) 严重程度低的优先级不一定低。如果软件的界面不是很方便用户使用或软件的名字对公司的形象有一定的影响，这样的缺陷虽然不是很严重，但却关系到软件和公司的市场形象，需要立即进行修正。

4) 按缺陷的起源和来源划分

软件缺陷的产生不仅仅是因为编程错误，更多的是因为在软件开发的初期做了错误或不全面的需求分析和系统设计而引起的，因此根据产生缺陷的起源和来源可以划分成 5 类：Requirement、Architecture、Design、Code 和 Test。

缺陷起源指的是在需求、系统架构、设计、编码以及测试等不同阶段发现的缺陷。

缺陷来源是指缺陷所在的地方，如文档、代码等。例如：Requirement，需求规格说明中的错误，或需求规格说明书不清楚而引起的缺陷；Architecture，由于构架定义或设计以及接口定义或设计的问题引起的缺陷；Design，指设计文档描述不准确、和需求规格说明不一致引起的缺陷；Code，由于编码出错而引起的缺陷；Test，测试不彻底、不全面而遗留的缺陷。

按照缺陷的来源对软件缺陷进行分类可以明确缺陷处理的负责人。

5) 按缺陷的根源划分

缺陷根源是指造成各种缺陷/错误的根本因素，以寻求软件开发流程的改进、管理水平的提高，包括：

(1) 测试策略：错误的测试范围，误解了测试目标，超越测试能力的目标等。

(2) 过程、工具和方法：无效的需求收集过程，过时的风险管理过程，不适用的项目管理方法，没有估算规程，无效的变更控制过程等。

(3) 团队/人：项目团队职责交叉，缺乏培训。没有经验的项目团队，缺乏士气和动机不纯等。

(4) 缺乏组织和通信：缺乏用户参与，职责不明确，管理失败等。

(5) 其他，例如：硬件，处理器缺陷导致算术精度丢失，内存溢出等；软件，操作系统错误导致无法释放资源，工具软件的错误，编译器的错误，“千年虫”问题等；工作环境，组织机构调整，预算改变，罢工，噪音，中断，工作环境恶劣。

6) 按照缺陷的生命周期划分

bug(缺陷)在英语中指的就是虫子，因此我们可以把缺陷看作有生命的小虫子，每一个缺陷都有一个从出生到死亡的周期，因此根据缺陷的生命周期可以这样划分：new、confirmed、fixed、closed、reopen 等。

每一个缺陷都是由测试人员发现并提交的，这个状态标注为 new(新建)；缺陷被提交后，由相应的负责人进行接受，即 confirmed(确认)状态；相应的负责人解决了该缺陷后，该缺陷的状态就改为 fixed(解决)，并且将其发给测试人员进行回归测试，防止产生其他错误；测试人员对已解决的缺陷进行回归测试，如果确定已经解决，那么缺陷的状态就改为 closed(关闭)，否则就需要返还给该缺陷的负责人重新修正；有的缺陷在以前的版本中已经关闭，但是在新的版本中又重新出现，则需要将其状态改为 reopen(重新打开)。

缺陷不同，其表现形式及后果也不相同，在评审或测试过程中由于评审人员或测试人员的角度不同，对缺陷的认识也就不同，对缺陷的描述定义也就不会完全相同。例如在设计评审阶段，关注的是软件设计是否满足需求分析的要求、软件功能是否被清晰描述；在单元测试阶段，采取“白盒”测试发现代码中的缺陷；在功能测试阶段，关注的是相对独立的功能模块或相关联的部件；在系统测试阶段，测试的则是软件产品的整体等。

在缺陷确认分类过程中可以分析不同阶段的缺陷情况，与标准缺陷类型进行关联，并能确认其注入阶段。软件缺陷类型标准与缺陷检测阶段及关联注入阶段的情况如表 4-1 所示。

表 4-1　缺陷类型标准与软件测试阶段

缺陷类型标准		缺陷检测阶段				关联注入阶段
类型名称	描述	设计评审	单元测试	功能测试	系统测试	
功能	功能实现、全局数据，算法	需求满足性、数据库	功能、程序逻辑	程序错误	程序、数据库、遗漏需求	需求分析、设计
赋值	说明、重名、作用域、限制、初始化	函数说明	赋值	边界值	边界值、无效域	编码
接口/时序	过程调用和引用、函数调用、数据块共享、消息传递	过程接口	过程接口			设计
联编打包	变更管理、配置库、版本控制			打包问题	安装	集成(工具库、版本控制)
文档	注释、需求、设计类文档	设计说明	设计问题	手册问题、需求问题、设计问题	手册及联机帮助	所有阶段
用户界面/检查	人机交互、屏幕控制、出错信息、日志、输入/输出	检查	检查	报表格式、界面控制、权限错误、提示信息	界面控制	设计、编码
算法	算法、局部数据结构、逻辑、指针、循环、递归	算法	算法			设计
标准/语法	程序设计规范、编码标准、指令格式等	规范	语法、规范			设计、编码
性能	不满足系统可测的属性值：执行时间、处理速率等				性能	设计
环境	设计、编译、测试、其他支持系统问题			测试环境	安装	集成、运行

缺陷分类在软件开发、软件测试以及各个软件阶段的评审中都得到了广泛应用。缺陷属性针对文档和代码具有不同的实用性，表 4-2 列出了各自适用范围。

表 4-2　缺陷分类适用范围

缺陷属性	软件测试	同行评审
缺陷标识(Identifier)	■	■
缺陷类型(Type)	■	■
缺陷严重程度(Severity)	■	■
解决优先级(Priority)	■	□
缺陷状态(Status)	■	□
缺陷起源(Origin)	■	■
缺陷原因(Cause)	●	●

■：需要记录　●：可以不考虑/可以记录　□：不考虑

4.1.4　软件缺陷管理

软件测试的任务是为了尽早发现软件系统中的缺陷，确保得以修复，并最终保证软件的质量。缺陷跟踪管理是软件测试中的一个有机组成部分，是 CMM 第二级的要求。在 CMM 第二级的软件组织中，软件项目从自身的需求出发，制定项目的缺陷管理过程。项目组将完整地记录开发过程中的缺陷，监控缺陷的修改过程，并验证修改缺陷的结果。

1. 缺陷管理的目标

软件缺陷能够引起软件运行时产生的一种不希望或不可接受的外部行为结果，软件测试过程简单说就是围绕缺陷进行的，对软件缺陷跟踪管理一般而言要达到以下目标：

(1) 确保每个被发现的缺陷都能够被解决。这里解决的意思不一定是被修正，也可能是其他处理方式(例如，在下一个版本中修正或不修正)。总之，对每个被发现缺陷的处理方式必须能够在开发组织中达成一致。

(2) 收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段。决定测试过程是否结束有很多种方式，通过缺陷趋势曲线来确定测试过程是否结束是常用且较为有效的一种方式。

(3) 收集缺陷数据并在其上进行数据分析，作为组织的过程财富。在对软件缺陷进行管理时，必须先对软件缺陷数据进行收集，然后才能了解这些缺陷，并且找出预防和修复它们的办法，以及预防引入新的缺陷。

2. 缺陷记录日志

缺陷记录日志用于软件缺陷数据的收集，收集软件缺陷数据的步骤如下：

(1) 为测试和同行评审中发现的每一个缺陷做一条记录。

(2) 对每个缺陷要记录足够详细的信息，以便以后能更好地了解这个缺陷。

(3) 分析这些数据以找出哪些缺陷类型引起大部分的问题。

(4) 设计发现和修复这些缺陷的方法(缺陷排除)。

表 4-3 是缺陷记录日志。

表 4-3　缺陷记录日志

<table>
<tr><th>日期</th><th>编号</th><th>状态</th><th>类型</th><th>缺陷来源</th><th>排除来源</th><th>修改时间</th><th>修复时间</th></tr>
<tr><td rowspan="2"></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">描述</td></tr>
<tr><td rowspan="2"></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">描述</td></tr>
</table>

3. 软件缺陷管理流程

根据对国内外著名 IT 公司缺陷管理流程的研究，一般软件缺陷管理流程如图 4-3 所示：

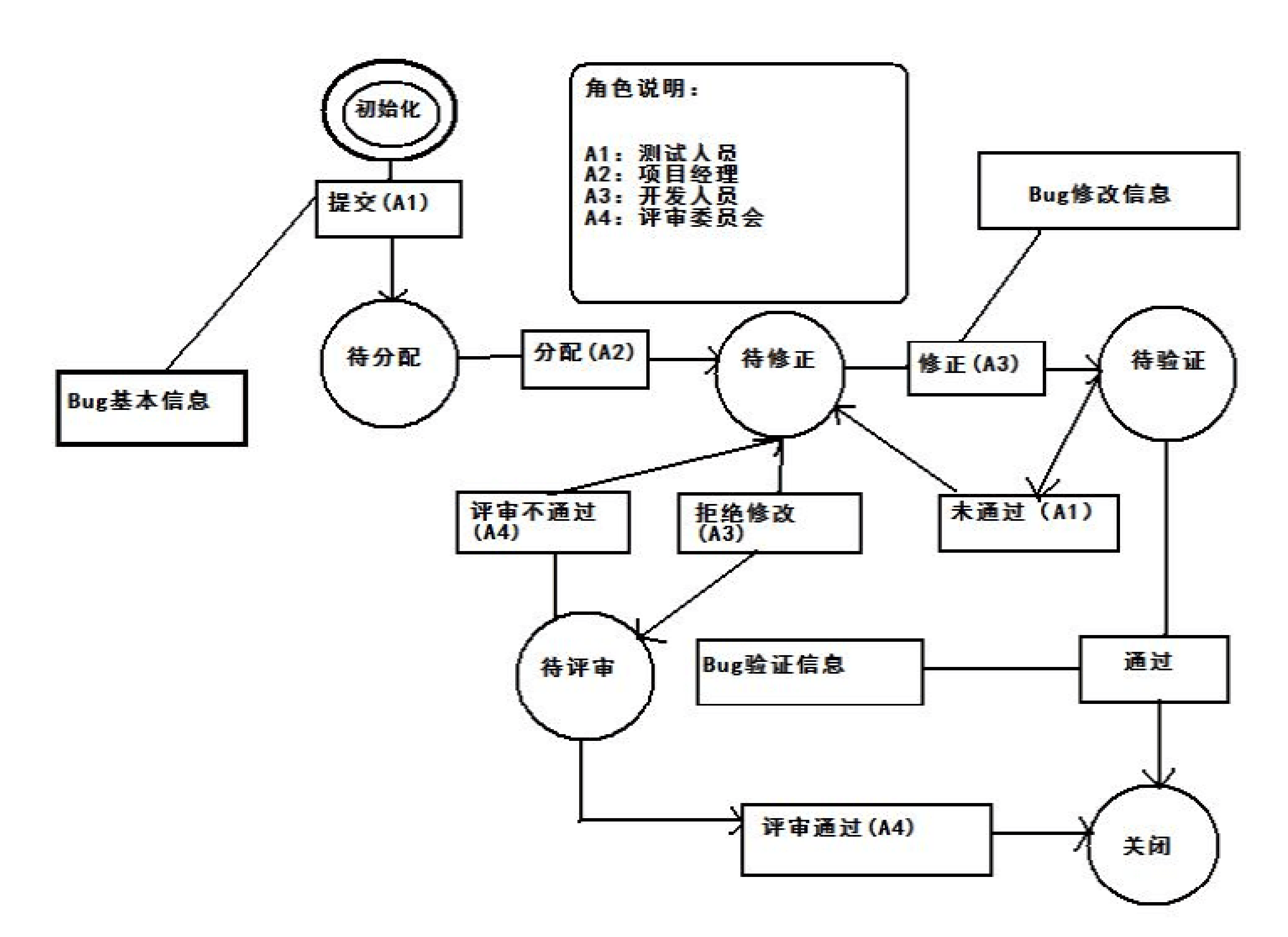


图 4-3　缺陷的一般管理流程

1) 缺陷管理流程中的角色

在缺陷管理流程中有四个角色：①测试人员 A1(进行测试的人员，并且是缺陷的发现者)；②项目经理 A2(对整个项目负责，对产品质量负责的人员)；③开发人员 A3(执行开发任务的人员，完成实际的设计和编码工作以及缺陷的修复工作)；④评审委员会 A4(对缺陷进行最终确认，在项目成员对缺陷达不成一致意见时，行使仲裁权利)。

2) 缺陷管理流程中的缺陷状态

在缺陷管理流程中包含 6 种缺陷：①初始化(缺陷的初始状态)；②待分配(缺陷等待分配给相关开发人员处理)；③待修正(缺陷等待开发人员修正)；④待验证(开发人员已完成修正，等待测试人员验证)；⑤待评审(开发人员拒绝修改缺陷，需要评审委员会评审)；⑥关闭(缺陷已被处理完成)。

软件缺陷管理流程描述如下：

(1) 测试小组发现新的缺陷，并记录缺陷，此时缺陷状态为“初始化”。

(2) 测试小组向项目经理提交新发现的缺陷(包括缺陷的基本信息)，此时缺陷的状态为“待分配”。

(3) 项目经理接收到缺陷报告后，根据缺陷的详细信息，确定处理方案，此时缺陷的状态为“待修正”。

(4) 缺陷报告被分配给特指的开发人员，开发人员对缺陷进行修复，并填写缺陷的修改信息，然后等待测试人员对修复后的缺陷再一次进行验证，此时缺陷的状态为“待验证”。

(5) 经测试人员验证后，发现缺陷未被修复，则重新交给原来负责修复的开发人员，转 3)，测试缺陷的状态为“待修正”。

(6) 经测试人员验证后，发现缺陷被修复，则填写缺陷验证信息，缺陷修复完成，此时缺陷的状态为“关闭”。

(7) 若测试人员验证缺陷未被修复，但是开发人员认为已修复完成拒绝再次修复，则将缺陷报告提交给评审委员会，等待评审委员会的评审，此时缺陷的状态为“待评审”。

(8) 若评审委员会评审不通过，即软件缺陷未被修复，开发人员需要继续修复，此时

软件缺陷的状态为“待修正”。

(9) 若评审委员会评审通过，即软件缺陷被修复，此时缺陷的状态为“关闭”。

4) 实施缺陷管理流程的注意事项

在整个缺陷的跟踪管理流程中，为了保证所发现的错误是真正的错误，需要有丰富测试经验的测试人员验证和确认发现的缺陷是否为真正的缺陷，发现的缺陷是由什么引起的，以及测试步骤是否准确、简洁、可以重复等。除此之外，由于对软件设计具体要求的不了解，对测试报告中的个别软件错误可能无法确认是否属于真正的软件错误，本地化服务商需要与软件供应商交流并确认；对于缺陷的处理都要保留处理信息，包括处理者姓名、时间、处理方法、处理步骤、错误状态、处理注释等；对缺陷的拒绝不能由程序员单方面决定，应该由项目经理、测试经理和设计经理组成的评审委员会决定；缺陷修复后，必须由报告缺陷的测试人员验证并确认已经修复，才能关闭缺陷。另外在缺陷跟踪管理流程中，还应注意以下几点：

(1) 测试小组在提交事务时，应清楚详细地将问题描述出来，便于项目经理进行处理。

(2) 项目经理在确定处理方案时，若对测试小组提出的事务有疑问，应及时与测试小组人员沟通，以保证完全理解测试小组提出的事务，确定正确的处理方案。同样，缺陷修复人员在处理事务时，若对测试小组提出的事务有疑问，也应及时与测试小组人员沟通，以保证准确处理测试小组提出的事务。

(3) 修复人员在解决事务时，应将发现原因、解决的途径和方法详细地描述出来，以便日后查阅。

(4) 测试小组成员应定期整理并归类测试的 bug，并写成测试报告，向项目经理、技术总监报告测试结果。

4.2　软件缺陷度量、分析与统计

在软件开发过程中实施缺陷的度量与分析对于提高软件开发和测试效率、预防缺陷发生、保证软件产品质量有着十分重要的作用。另外，对软件的缺陷进行跟踪管理的目标之一是对缺陷的数据进行统计。通过对软件开发过程中发现的缺陷进行分析统计，可以判断软件质量、项目的进展。

4.2.1　软件缺陷度量

缺陷度量就是对项目过程中产生的缺陷数据进行采集和量化，将分散的缺陷数据统一管理，使其有序而清晰，然后通过采用一系列数学函数，对数据进行处理，分析缺陷密度和趋势等信息，从而提高产品质量和改进开发过程。缺陷度量是软件质量度量的重要组成部分，它和软件测试密切相关。尽管缺陷度量本身并不能发现缺陷、剔除缺陷，但是有助于这些问题的解决。

软件缺陷度量的方法较多，从简单的缺陷计数到严格的统计建模，其主要的度量方法有缺陷密度(软件缺陷在规模上的分布)、缺陷率(缺陷在时间上的分布)、整体缺陷清除率、

阶段性缺陷清除率、缺陷趋势、预期缺陷发现率等。

1. 缺陷密度

Myers 有一条关于软件测试的著名的反直觉原则：在测试中发现缺陷多的地方，还有更多的缺陷将会被发现。这条原则背后的原因在于：在发现缺陷多的地方、漏掉的缺陷可能性也会越大，或者告诉我们测试效率没有被显著改善之前，在纠正缺陷时将引入较多的错误。这条原则的数学表达就是缺陷密度的度量——每 KLOC 或每个功能点(或类似功能点的度量——对象点、数据点、特征点等)的缺陷数，缺陷密度越低意味着产品质量越高。

如果缺陷密度跟上一个版本相同或更低，就应该分析当前版本的测试效率是不是降低了？如果不是，意味着质量的前景是乐观的；如果是，那么就需要额外的测试，还需要对开发和测试的过程进行改善。

如果缺陷密度比上一个版本高，那么就应该考虑在此之前是否为显著提高测试效率进行了有效的策划并在本次测试中得到实施？如果是，虽然需要开发人员做更多的努力去修正缺陷，但质量还是得到了更好的保证；如果不是，意味着质量恶化、质量很难得到保证。这时，要保证质量，就必须延长开发周期或投入更多的资源。

对于一个软件工作产品，软件缺陷分为两种：通过评审或测试等方法发现的已知缺陷(known defect)、尚未发现的潜在缺陷(latent defect)。缺陷密度的定义如下：

缺陷密度= 检测缺陷的数量/产品规模

在缺陷密度的公式中，产品规模的度量单位可以是文档页、代码行、功能点。缺陷密度是软件缺陷的基本度量，可用于设定产品质量目标、支持软件可靠性模型(如 Rayleigh 模型)、预测潜伏的软件缺陷，进而对软件质量进行跟踪和管理、支持基于缺陷计数的软件可靠性增长模型(如 Musa-Okumoto 模型)、对软件质量目标进行跟踪并评判能否结束软件测试。

2. 缺陷率

缺陷率的通用概念是一定时间范围内的缺陷数与错误概率(OFE，Opportunities For Error)的比值。前面我们已经讨论过软件缺陷和失败的定义，失败是缺陷的实例化，可以用观测到的失败的不同原因数目来近似估算软件中的缺陷数目。

软件产品的缺陷率，即使对于一个特定的产品，在其发布后的不同时段也是不同的。例如，从应用软件的角度来说，90%以上的缺陷是在发布后两年内被发现的，而对于操作系统，90%以上的缺陷通常在产品发布后需要四年的时间才能被发现。

3. 缺陷清除率

缺陷清除率(亦叫“缺陷排除率”)，英文缩写 DER(Defect Elimination Rate)，可以用于缺陷的预测和分析。

DER 分为两种：整体缺陷清除率和阶段性缺陷清除率，阶段性缺陷清除率能够反映整体缺陷清除率。

缺陷清除率=检测缺陷/所有缺陷

由于潜在的缺陷不容易确定，因此：

缺陷清除率≈检测缺陷/(检测缺陷+以后发现的缺陷)

1) 整体缺陷清除率

先引入几个变量，F 为描述软件规模用的功能点；D1 为在软件开发过程中发现的所有缺陷数；D2 为软件发布后发现的缺陷数；D 为发现的总缺陷数。因此，D=D1+D2。

对于一个应用软件项目，有如下计算方程式(从不同的角度估算软件的质量)：

质量 ＝D2/F

缺陷注入率 ＝D/F

整体缺陷清除率= D1/D

假如有 100 个功能点，即 F=100，而在开发过程中发现了 20 个错误，提交后又发现了 3 个错误，则有 D1 =20，D2 =3，D= D1+D2 =23。

质量(每功能点的缺陷数)=D2/F= 3/100= 0.03 (3%)

缺陷注入率= D/F= 20/100= 0.20 (20%)

整体缺陷清除率= D1/D= 20/23= 0.8696 (86.96%)

有资料统计，美国的平均整体缺陷清除率目前只达到大约 85%，而对于一些具有良好管理和流程等的著名软件公司，其主流软件产品的缺陷清除率可以超过 98%。

众所周知，清除软件缺陷的难易程度在各个阶段也是不同的。需求错误、规格说明、设计问题及错误修改是最难清除的，如表 4-4 所示。

表 4-4　不同缺陷源的清除率

缺陷源	潜在缺陷	清除率(%)	被交付的缺陷
需求报告	1.00	77	0.23
设计	1.25	85	0.19
编码	1.75	95	0.09
文档	0.60	80	0.12
错误修改	0.40	70	0.12
合计	5.00	85	0.75

表 4-5 反映的是 CMM 五个等级是如何影响软件质量的，其数据来源于美国空军 1994 年委托 SPR(美国一家著名的调查公司)进行的一项研究。从表 4-5 中可以看出，CMM 级别越高，缺陷清除率也越高。

表 4-5　SEI CMM 级别的潜在缺陷与清除率

SEI CMM 级别	潜在缺陷	清除效率(%)	被交付的缺陷
1	5.00	85	0.75
2	4.00	89	0.44
3	3.00	91	0.27
4	2.00	93	0.14
5	1.00	95	0.05

2) 阶段性缺陷清除率

阶段性缺陷清除率是测试缺陷密度度量的扩展。除测试外，还要求跟踪开发周期所有阶段中的缺陷，包括需求评审、设计评审、代码审查。因为编程缺陷中的很大百分比同设

计问题有关，所以进行正式评审或功能验证以增强前期过程的缺陷清除率有助于减少出错的注入。阶段的缺陷清除模型在某种程度上能够反映开发工程总的缺陷清除能力。

下面是缺陷清除有效性(DRE，Defect Remove Efficiency)的分析，DRE 可以定义为：

DRE=开发阶段清除的缺陷数/产品潜伏的缺陷总数*100％

因为产品缺陷的总数是不知道的，所以必须通过一些方法获得其近似值，如经典的种子公式方法，或近似等于当前阶段清除的缺陷数+以后发现的缺陷数。当 DRE 用于前期阶段和特定阶段时，此时 DRE 相应地被称为早期缺陷清除有效性和阶段有效性，对给定阶段的残留缺陷数，可以估计为：

当前阶段的潜伏缺陷数 = 本阶段发现的缺陷数/(本阶段入口存在的缺陷数+本阶段注入的缺陷数)*100％

给定阶段的 DRE 度量值越高，遗漏到下一阶段的缺陷数就越少。

缺陷是在各个阶段注入阶段性产品或成果的，通过表 4-6 描述的与缺陷注入和清除相关联的活动分析，可以更好地理解缺陷清除有效性。回归缺陷是由于修正当前缺陷时而引起的相关的、新的缺陷，所以即使在测试阶段，也会产生新的缺陷。

表 4-6　与缺陷注入和清除相关联的活动

开发阶段	缺陷注入	缺陷清除
需求	需求收集过程和功能规格说明书	需求分析和评审
系统/概要设计	设计工作	设计评审
详细/程序设计	设计工作	设计评审
编码和单元测试	编码	代码审查、测试
集成测试	集成过程、回归缺陷	构建验证、测试
系统测试	回归缺陷	测试、评审
验收测试	回归缺陷	测试、评审

清除的缺陷数等于检测到的缺陷数减去不正确修正的缺陷数+未修正的缺陷数。如果不正确修正的缺陷数+未修正的缺陷数所占的比例很低(经验数据表明，测试阶段大概为2%)，清除的缺陷数就近似于检测缺陷数。

4. 缺陷趋势

“趋势”一词在词典里的解释是“事物发展的动向”，也就是会呈现出某种规律，可用来对未来进行预测。缺陷趋势是在一定周期时间或一定阶段内，产生/发现缺陷的动向或规律，是缺陷率按时间或按阶段增长/下降的动态分布。周期可以是天、周、月。阶段可以是版本，例如 1.2、1.3、1.4。通常缺陷趋势用缺陷趋势图表示。

5. 预期缺陷发现率

缺陷发现率，英文缩写为 DDR(Defect Discovery Rate)，描述在特定时间段内发现缺陷数目的一种度量，常常以图表形式来显示。计算方法是计算测试人员各自发现的缺陷数总和除以各自所花费的测试时间总和。

缺陷发现率=∑提交缺陷数(个)/∑执行测试的有效时间(小时)

预期缺陷发现率则是通过对缺陷发现率的分析，预期在将来的某段时间内可能发现的缺陷数。

$$预期缺陷发现率=\sum可能发现的缺陷数(个)/\sum未来的某段时间内(小时)$$

许多组织都把缺陷发现率当作一个帮助自己判断测试是否可以结束、预测产品发布日期的重要度量。

虽然缺陷发现率趋势下降一般都是不错的信息，但是我们必须提防其他可能导致发现率下降的因素(工作量减少、没有新的测试用例等)，所以我们的重要决策往往要依据多个支撑性度量元。

正常来讲，到测试后期，每天发现的新缺陷的数量呈下降趋势。如果我们假定每天工作量是恒定的，那么每发现一个缺陷所消耗的成本也会呈上升趋势。到某个点后，继续进行测试，发现缺陷所需成本已经超过缺陷本身带来的损失。这时候测试可以退出了。

4.2.2　软件缺陷分析

缺陷分析是对软件开发各个阶段产生的缺陷信息进行分类和汇总统计、计算分析指标、编写分析报告的活动。

1. 缺陷分析的意义

通过软件缺陷分析可以发现各种类型缺陷发生的概率，掌握缺陷集中的区域、明确缺陷发展趋势、挖掘缺陷产生的根本原因，便于有针对性地提出遏制缺陷发生的措施、降低缺陷数量。缺陷分析报告中的统计数据及分析指标既是对当前软件质量状况的评估，也是判定软件是否能按期发布或交付使用的重要依据。实施缺陷分析的前提是需要一个符合项目要求的缺陷数据管理系统，通过采集完整的缺陷数据信息，进行缺陷数据分析，从而改进软件过程质量并实施缺陷预防措施。

缺陷分析也可以用来评估当前软件的可靠性，并且预测软件产品的可靠性变化，缺陷分析在软件可靠性评估中占有相当大的作用。

另外，通过缺陷分析达到缺陷预防的目的，这是缺陷管理的核心任务之一。

2. 缺陷预防

缺陷预防的着眼点在于寻找缺陷的共性原因。通过寻找、分析和处理缺陷的共性原因，实现缺陷预防。缺陷预防并不是一个不切实际的目标，测试人员在开发过程中应该积极为开发小组提供缺陷分析，就有可能降低缺陷产生的数量。因此，缺陷管理的最终目标是预防缺陷，不断提高整个开发团队的技能和实践经验，而不只是修正它们。

缺陷预防策略非常简单和容易实现，策略是：①测试活动尽量提前，通过及时消除开发前期阶段引入的缺陷，防止这些缺陷遗留并放大到后续环节；②通过对已有缺陷进行分析，找出产生这些缺陷的技术上的不足和流程上的不足(缺陷的根源)，然后寻找一个方法来对这些不足进行改进，预防类似的缺陷在将来出现。这种策略费用并不高，但能带来极大的好处。

3. 缺陷分析步骤

缺陷分析的第一步是记录缺陷，值得注意的是：记录缺陷不应该满足于记录缺陷的表面症状。测试的一个重要职责就是试图发现缺陷的根本原因，在测试时不应将产品看作一个黑盒，而应该像开发人员那样了解产品的内在，包括深入源代码、理解产品的设计和实现。

缺陷分析的第二步是对测试出来的缺陷进行缺陷分类，找出那些关键的缺陷类型，进一步分析其产生的根源，有针对性地制定改进措施。缺陷分析非常关键的一步就是寻找一个预防类似缺陷再次发生的方法。这一方法不仅涉及开发人员、测试人员，还涉及不直接负责代码编写的资深开发人员。利用这一阶段的实践成果，开发人员可以预防缺陷的发生，而不仅仅是修正这些缺陷。

缺陷分析的第三步是进行缺陷预防分析，它是整个缺陷分析过程的核心。这一阶段总结出的实践可以在更广泛的范围内预防潜在的缺陷。由于分析结果的广泛应用性，分析某个具体缺陷的投入将很容易被收回。这个时候，缺陷分析提供了两个非常重要的参数，一个是缺陷数量的趋势，另一个是缺陷修复的趋势。缺陷趋势就是将每月新生成的缺陷数、每月被解决的缺陷数和每月遗留的缺陷数标记成一张趋势图表。

一般在项目的开始阶段，会发现缺陷数曲线会呈上升趋势，到项目中后期被修复缺陷数曲线会趋于上升，而发现缺陷数曲线应总体趋于下降。同时处于 Open 状态的缺陷也应该总体呈下降趋势，到项目最后，三条曲线都趋向于零。项目经理可通过持续观察这张图表，确保项目开发健康发展。同时，通过分析、预测项目测试缺陷趋于零的时间，以制定产品质量验收和发布的时间。

缺陷分析的最后一步是编写缺陷分析报告，绘制缺陷分析图。

缺陷分析报告中的统计数据及分析指标既是对软件质量的权威评估，也是确定测试是否达到结束标准、判定测试是否已达到客户可接受状态和判定软件是否能发布或交付使用的重要依据。

另外，缺陷分析图表会告诉我们很多有价值的信息。比如，可分析开发和测试在人力资源的配比上是否恰当，可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常波动，如本来应该越测试越收敛的，却到了某个点发现的故障数反而呈上升趋势，那么意味着往往有一些特殊事件的发生。通过对测试缺陷进行分析，能够给予我们很多改进研发和测试工作的信息。

4. 缺陷分析方法

在缺陷分析中，常用的主要缺陷参数有四个：①状态，缺陷的当前状态(打开的、正在修复或关闭的等)；②优先级，必须处理和解决缺陷的相对重要性；③严重性，缺陷的相关影响，对最终用户、组织或第三方的影响，等等；④起源，导致缺陷的起源故障及其位置，或排除该缺陷需要修复的构件。

可以将缺陷计数作为时间的函数来报告，即创建缺陷趋势图或报告；也可以将缺陷计数作为一个或多个缺陷参数的函数来报告，如作为缺陷密度报告中采用的严重性或状态参数的函数。这些分析类型分别为揭示软件可靠性的缺陷趋势或缺陷分布提供了判断依据。

例如，预期缺陷发现率将随着测试进度和修复进度而最终减少。可以设定一个阈值，在缺陷发现率低于该阈值时才能部署软件。也可根据执行模型中的起源报告缺陷计数，以

允许检测“较差的模块”、“热点”或需要再三修复的软件部分，从而指示一些更基本的设计缺陷。

这种分析中包含的缺陷必须是已确认的缺陷。不是所有已报告的缺陷都报告实际的缺陷，这是因为某些缺陷可能是扩展请求，超出了项目的规模，或描述的是已报告的缺陷。然而，需要查看并分析一下，为什么许多报告的缺陷不是重复的缺陷就是未经确认的缺陷，这样做是有价值的。

国内外进行缺陷分析常用如下几种方法：

(1) ODC 缺陷分析：由 IBM 的 Waston 中心推出。Phontol.com 将一个缺陷在生命周期各环节的属性组织起来，从单维度、多维度来对缺陷进行分析，从不同角度得到各类缺陷的缺陷密度和缺陷比率，从而积累得到各类缺陷的基线值，用于评估测试活动、指导测试改进和整个研发流程的改进；同时根据各阶段缺陷分布得到缺陷去除过程特征模型，用于对测试活动进行评估和预测。Phontol.com 上面回答中涉及的缺陷分布、缺陷趋势等都属于这个方法中的一个角度而已。

(2) Gompertz 分析：根据测试的累计投入时间和累计缺陷增长情况，拟合得到符合自己过程能力的缺陷增长 Gompertz 曲线，用来评估软件测试的充分性、预测软件极限缺陷数和退出测试所需时间、作为测试退出的判断依据、指导测试计划和策略的调整。

(3) Rayleigh 分析：通过生命周期各阶段缺陷发现情况得到缺陷 Rayleigh 曲线，用于评估软件质量、预测软件现场质量。

(4) 四象限分析：根据软件内部各模块、子系统、特性测试所累积时间和缺陷去除情况，与累积时间和缺陷去除情况的基线进行比较，得到各个模块、子系统、特性测试分别所位于的区间，从而判断哪些部分测试可以退出、哪些测试还需要加强，用于指导测试计划和策略的调整。

(5) 根本原因分析：利用鱼骨图、柏拉图等分析缺陷产生的根本原因，根据这些根本原因采取措施，改进开发和测试过程。

(6) 缺陷注入分析：对被测软件注入一些缺陷，通过已有用例进行测试，根据这些刻意注入缺陷的发现情况，判断测试的有效性、充分性，预测软件残留缺陷数。

(7) DRE/DRM 分析：通过已有项目历史数据，得到软件生命周期各阶段缺陷注入和排除的模型，用于设定各阶段质量目标，评估测试活动。

4.2.3　软件缺陷统计

软件缺陷统计是软件分析报告中的重要内容之一。事实上，从统计的角度出发，可以对软件过程的缺陷进行度量，如软件功能模块缺陷分布、缺陷严重程度分布、缺陷类型分布、缺陷率分布、缺陷密度分布、缺陷趋势分布、缺陷注入率/消除率等。统计的方式可以用表格，也可用图表表示，如散点图、趋势图、因果图、直方图、条形图、排列图等。

1. 软件功能模块的缺陷统计

图 4-4 中的软件包含文件、编辑、视图、插入、格式、工具、帮助等功能模块。

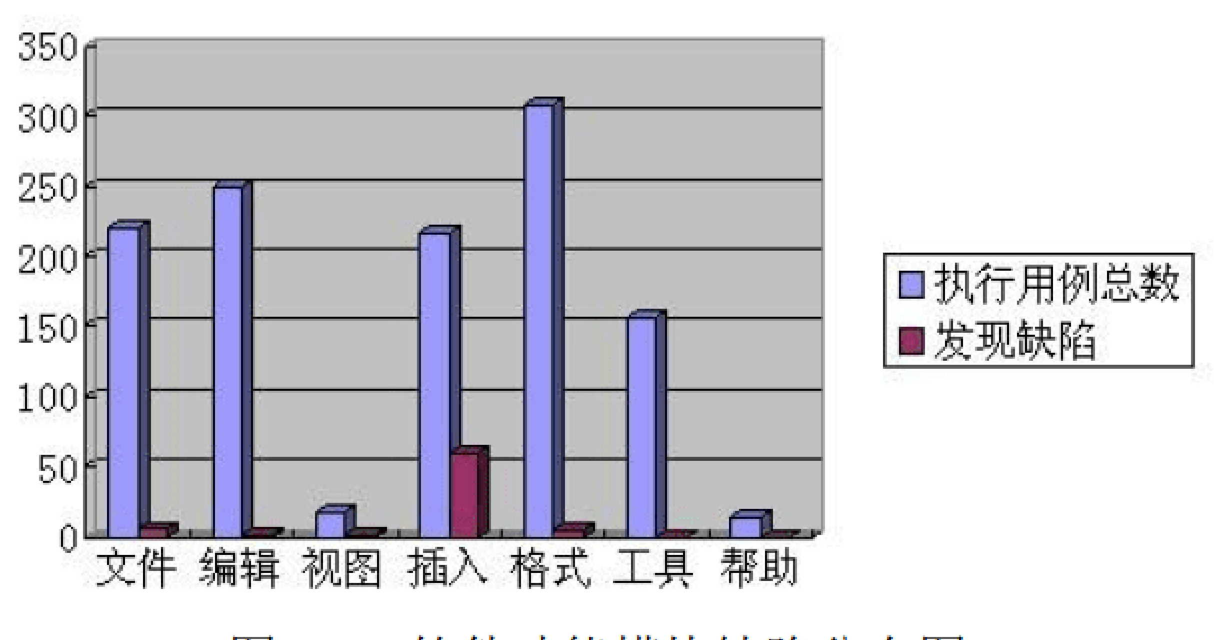


图 4-4　软件功能模块缺陷分布图

图 4-4 说明，在某些模块，执行的测试用例多，但是没有成比例地发现很多缺陷，所以这些模块是比较成熟的，因为这些模块几乎不怎么修改，再测试的话，也不会发现什么问题的；但是某些模块执行的测试少，却发现了更多的缺陷，这些模块修复的地方，或者发生功能变更的可能性大，所以将成为质量不稳定的关键点。并且在以后的回归测试中，应该在质量不稳定的模块中投入更多的人手和时间，进行更全面的测试，其他模块就相应减少测试工作的投入。这样，测试工作的压力就不是那么大了，而且效率也会相对提高。

2. 缺陷严重程度统计

按照缺陷严重程度及阶段缺陷分布可以统计整个项目生命周期中所有同行评审的缺陷分布，也可以统计某一阶段所有同行评审的缺陷分布，如图 4-5 所示。

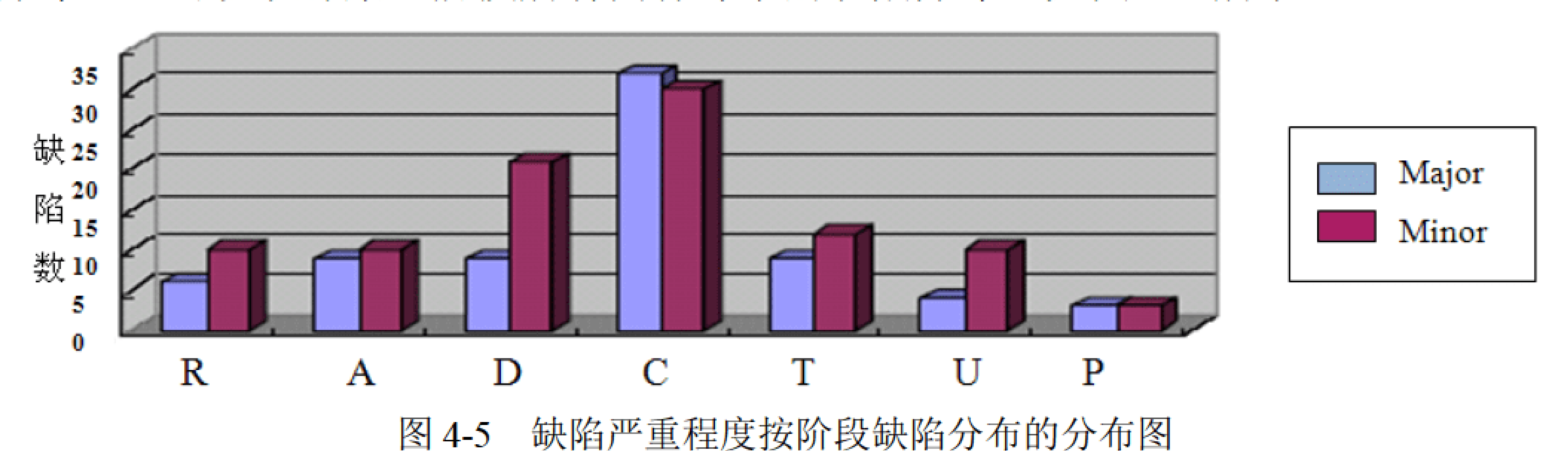


图 4-5　缺陷严重程度按阶段缺陷分布的分布图

多个项目按阶段缺陷率的统计分布如图 4-6 所示。

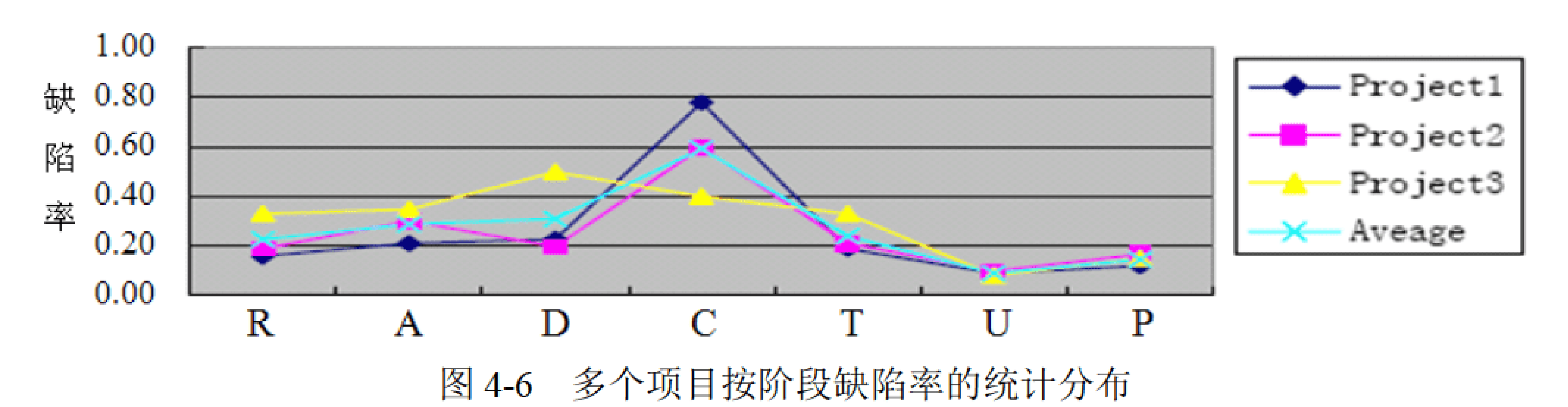


图 4-6　多个项目按阶段缺陷率的统计分布

3. 缺陷类型统计

按照缺陷类型统计分布图，可以是某一次评审的缺陷统计，可以是某一类型缺陷评审的缺陷统计，也可以是某一阶段所有同行评审的缺陷统计，还可以是整个项目周期内所有同行评审的缺陷统计，如图 4-7 所示。

我们可以根据图 4-7 找出那些关键的缺陷类型，进一步分析其产生的根源，从而有针对性地制定改进措施。图 4-8 展示的是某软件系统测试的缺陷类型饼图。

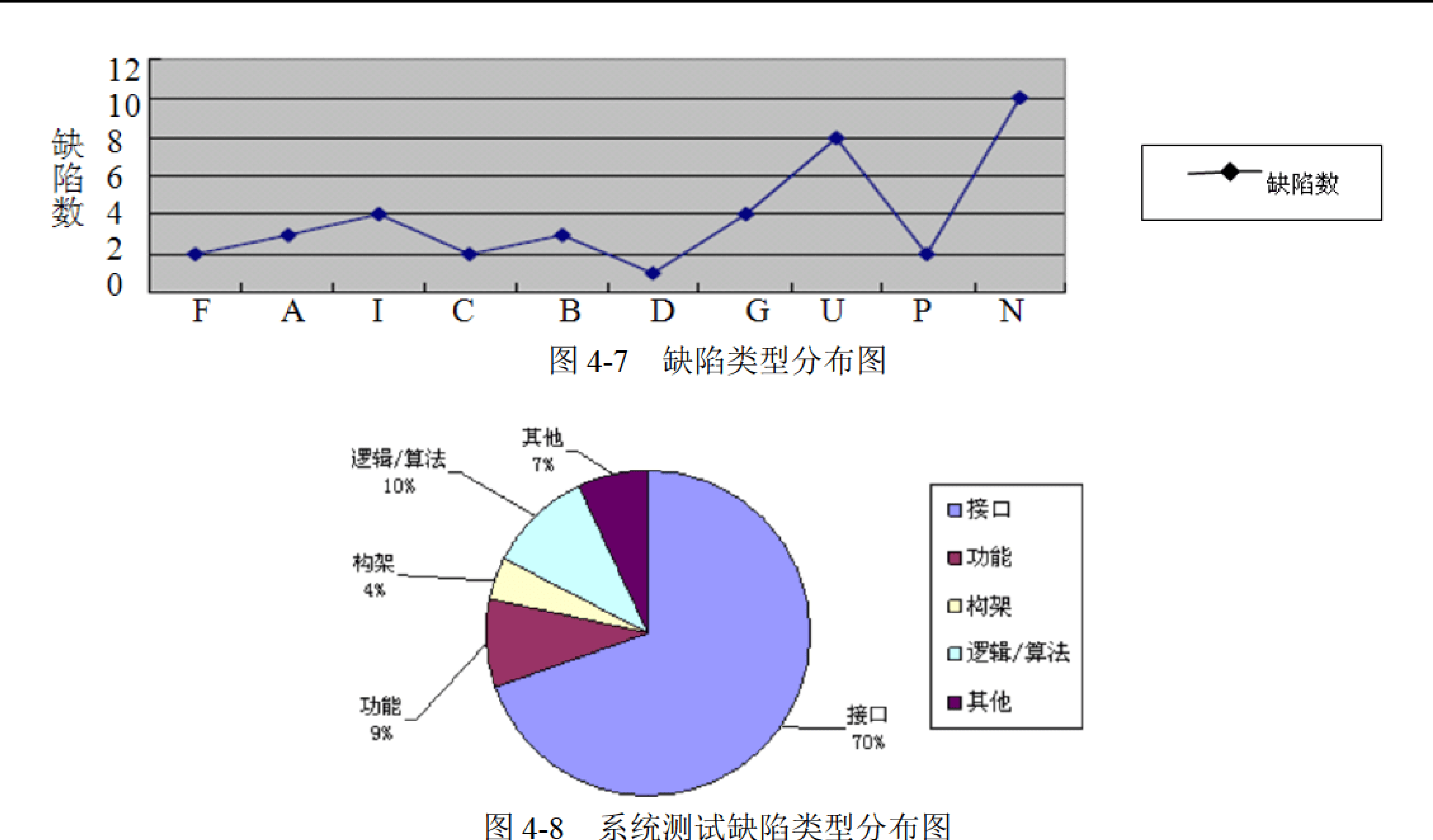


图 4-7　缺陷类型分布图

图 4-8　系统测试缺陷类型分布图

从系统测试故障来看，有较多故障是由接口原因造成的，细分有以下几种原因：

(1) 跨项目间的接口，接口设计文档的更改没有建立互相通知的机制，导致接口问题到系统测试时才暴露。

(2) 部门内部跨子系统的接口，由于本项目设计文档是按功能规划而不是按照产品组件编写的，一般由主要承接功能工作的组编写该文档，接口内容可能不为其他开发组理解并熟悉，导致因接口问题而出错。

(3) 系统设计基线化后，更改系统接口，没有走严格的变更流程，进行波及分析，导致该接口变更只在某个子系统中被修改，而使错误遗漏下来。

为此，我们可以有针对性地制定改进建议：

(1) 对接口文档的评审，一定要识别受影响的相关干系人，使他们了解并参与接口设计的把关。

(2) 对基线化的接口设计文档的变更一定要提交变更单给变更控制委员会(CCB)决策，并做好充分的波及影响分析，以便同步修改所有关联的下游代码。

(3) 概要设计文档按子系统规划，详细设计文档按模块规划，通过相关组参加评审协调接口设计。

以上缺陷分类只是为了描述方便，本身描述并不尽合理。实际定义缺陷分类可能有多个维度，如发现活动、引入活动、缺陷来源、缺陷类型、严重程度等。只要满足自己的缺陷管理、缺陷分析需求即可。

4. 缺陷趋势图

项目管理中一项非常重要但也十分困难的工作是衡量项目的进度、质量、成本等，统称为项目的状态，以确定项目是否能按期保质完成。这方面，测试提供了两个非常重要的参数，一个是缺陷数量的趋势(如图 4-9 所示)，另一个是缺陷修复的趋势(提示：这里所说的测试均指代系统测试)。

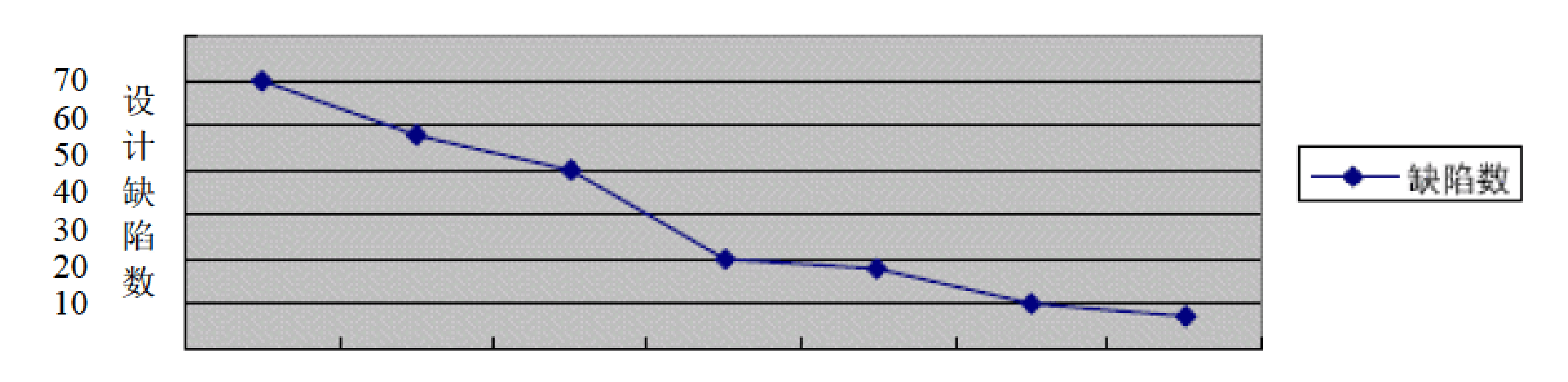


图 4-9　基于时间(周)的缺陷率趋势图

缺陷趋势就是将每月/每周新生成的缺陷数、每月被解决的缺陷数和每月/每周遗留的缺陷数标记成一张趋势图表。一般在项目的开始阶段被发现缺陷数曲线会呈上升趋势，到项目中后期被修复缺陷数曲线会趋于上升；而被发现缺陷数曲线应总体趋于下降；同时处于 Open 状态的缺陷数也应该总体呈下降趋势，到项目最后，三条曲线都趋向于零，如图 4-10 所示。

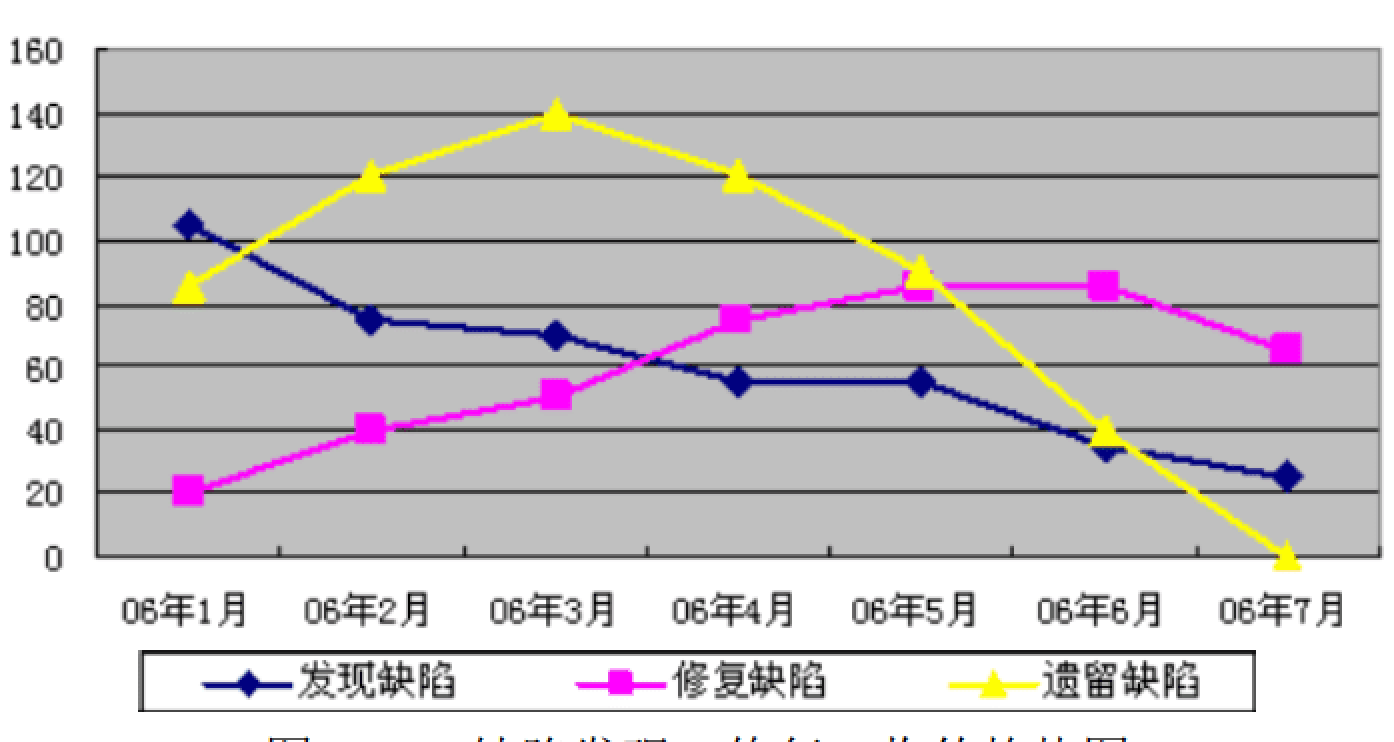


图 4-10　缺陷发现、修复、收敛趋势图

项目管理者会持续观察这张图表，确保项目健康发展，同时通过分析，预测项目测试缺陷数趋于零的时间。在一定的历史经验的基础上分析，使用这一图表会得到很多有价值的信息，比如，可分析开发和测试在人力资源的配比上是否恰当，可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常波动，如本来应该越测试越收敛的，却到了某个点，发现的故障数反而呈上升趋势，那么这些点往往有一些特殊事件发生。比如：在该时间段要进行回归测试的版本增加了新的功能，导致缺陷引入；该回归版本没有进行集成测试就直接进行最终的系统测试；等等。

当然，这个统计周期也可以根据我们的项目实施情况进行，比如按照回归版本的版本号进行统计、按周进行统计等。也有公司把缺陷收敛情况当作判断版本是否可以最终外发的标志。

项目的缺陷率按版本的趋势见图 4-11。

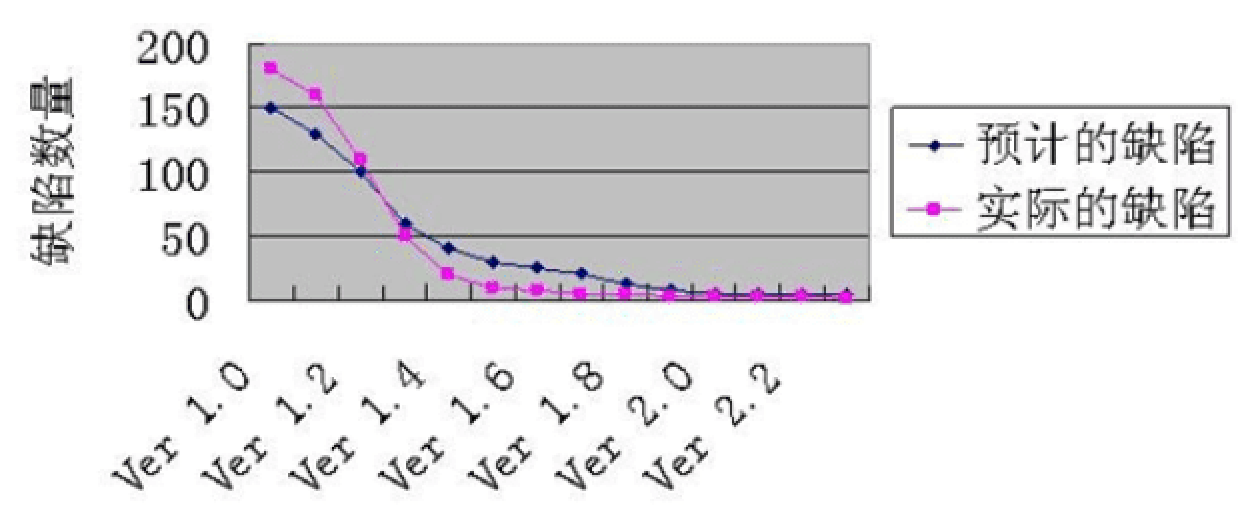


图 4-11　基于版本的缺陷率趋势图

这张缺陷趋势分析图说明软件在测试版本的 Ver 1.4 时，软件的质量已经得到了很好的控制，在 Ver 1.8 时，基本上就已经可以发布软件了，后面的测试几乎是没有什么意义的。原因很简单，软件中的缺陷既然是不可能全部发现的，就不要指望找出软件中全部的缺陷，当足够少(各公司的定义是不同的)时，就应该停止测试了。

4.3　软件缺陷报告

缺陷一旦被发现后，测试人员下一步要做的工作是就所发现的缺陷与开发人员进行沟通，最简单的沟通方法就是通过缺陷报告。缺陷报告的重要性不仅仅在于文档化，同时也是使发现的缺陷被修正的唯一方法；另外，缺陷报告记录了缺陷发生的环境，如各种资源的配置情况、缺陷的再现步骤以及缺陷性质的说明，更重要的是还记录着缺陷的处理过程和状态。而缺陷的处理进程从一定角度反映了测试的进程和被测软件的质量状况以及改善过程；最后，缺陷报告也是测试人员测试工作的最可见的产品，如果缺陷报告不能做到清晰、完整和易理解，并且缺陷不能重现，缺陷被拒绝修改或者开发人员无法重现缺陷的概率就会大大增加。

在软件测试过程中，每发现一个软件错误(缺陷)，都要记录该错误的特征和复现步骤等信息，以便相关人士分析和处理软件错误。为了便于管理测试发现的软件错误，通常要采用软件缺陷数据库，并将每一个发现的错误输入到软件缺陷数据库中。软件缺陷数据库中的每一条记录称为一份软件缺陷报告。

提供准确、完整、简洁、一致的缺陷报告是体现软件开发、测试与管理的专业性、高质量的主要评价指标。每份软件缺陷报告只书写一个缺陷或错误，这样可以每次只处理一个确定的错误，定位明确，提高效率，也便于修复错误后方便进行验证。

在缺陷的生命周期(缺陷从开始提出到最后完全解决，并通过复查的过程)中，缺陷报告的状态不断发生着变化，里面记录着缺陷的处理进程。

4.3.1　缺陷报告的主要内容

不同的项目和测试机构会依据不同的标准和规范来编制缺陷报告，目的是为缺陷报告阅读者识别缺陷提供足够的信息。

1. 缺陷报告的主要内容

一般情况下，缺陷报告主要包含下列内容：

(1) 问题报告编号：为了便于对缺陷进行管理，每个缺陷必须被赋予一个唯一的编号，编号规则可根据需要和管理要求制定。

(2) 标题：标题用简明的方式传达缺陷的基本信息，标题应该简短并尽量做到唯一，以便在观察缺陷列表时，可以很容易地注意到。

(3) 报告人：缺陷报告的原始作者，有时也可以包括缺陷报告的修订者。当负责修复缺陷的开发人员对报告有任何异议/疑义时，可以与报告人联系。

(4) 报告日期：首次报告的日期。让开发人员知道创建缺陷报告的日期是很重要的，

因为有可能这个缺陷在前面版本中曾经修改过。

(5) 程序(或组件)的名称：可分辨的被测试对象。

(6) 版本号：测试可能跨越多个软件版本，提供版本信息可以方便进行缺陷管理。

(7) 配置：发现缺陷的软件和硬件的配置，如操作系统的类型、是否有浏览器载入、处理器的类型和速度、RAM 的大小、可用的 RAM、正在运行的其他程序，等等。

(8) 缺陷的类型：如代码错误、设计问题、文档不匹配等。

(9) 严重性：描述所报告的缺陷的严重性。

(10) 优先级：由开发人员或管理人员进行确定，依据修复这个缺陷的重要性而定。

(11) 关键词：以便分类查找缺陷报告，关键词可在任何时候添加。

(12) 缺陷描述：对发现的问题进行详细说明，尽管描述要深入，但是简明仍是最重要的。缺陷描述的主要目的是说服开发人员决定修复这个缺陷。

(13) 重现步骤：这些步骤必须是有限的，并且描述的信息足够读者知道正确地执行就可以重现这个缺陷。

(14) 结果对比：在执行了缺陷重现步骤后，期望发生什么，实际上又发生了什么。

2. 报告缺陷的基本原则

在软件测试过程中，对于发现的大多数软件缺陷，要求测试人员简捷、清晰地把发现的问题报告提交给需要判断是否进行修复的小组，使其得到所需要的全部信息，然后才能决定怎么做。缺陷报告是测试人员的主要工作产品之一，好的缺陷报告会增加开发人员对测试人员的信任度，提高开发效率。因此，先来了解一下报告软件缺陷的基本原则。

(1) 尽快报告软件缺陷。软件缺陷发现越早，在软件开发过程中留下的修正时间越多，被修正的可能性越大，花费的代价就越少。图 4-12 显示了时间和缺陷修复之间的关系。

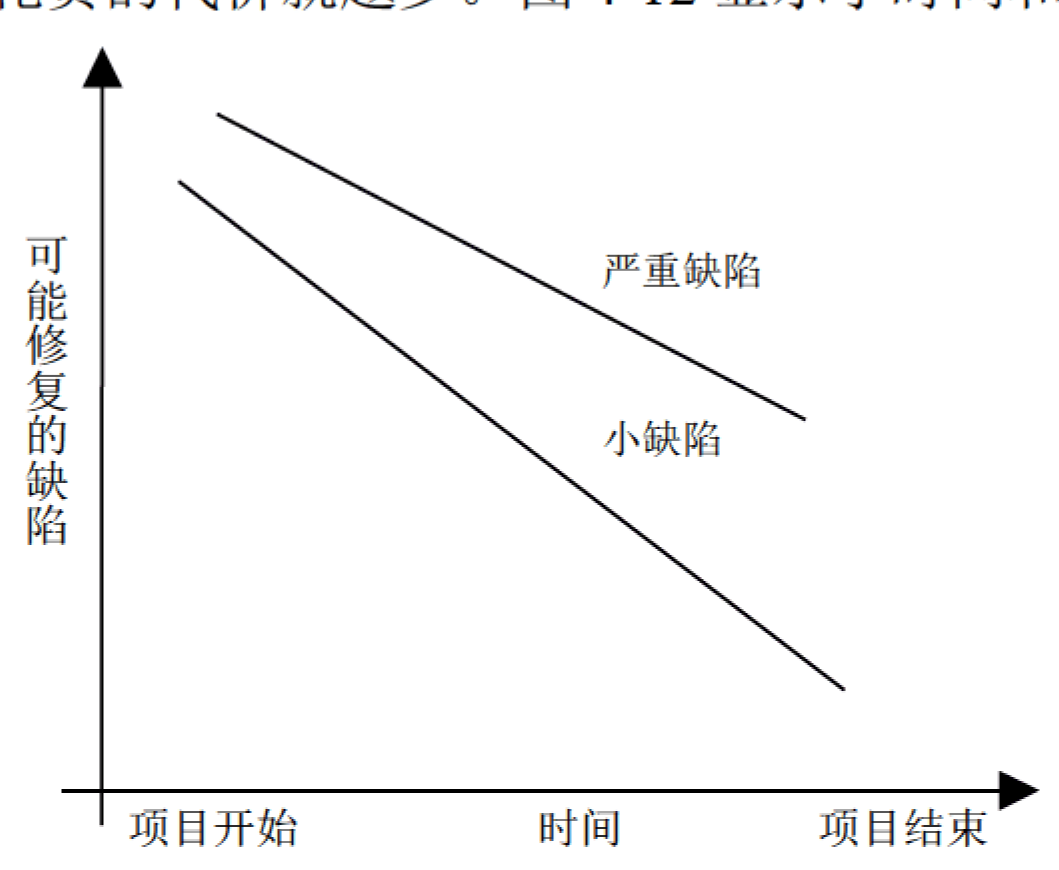


图 4-12　软件缺陷发现越晚，越不可能被修复

(2) 有效地描述软件缺陷。测试人员在提交缺陷报告时，应站在开发人员的角度思考问题，要确保开发人员拿到缺陷报告后马上就能明白问题，而不会产生理解上的歧义。假如你是开发人员，看到如下所述的报告：“无论何时在登录对话框中输入一串随机字符，软件就会进行一些奇怪的动作”。你肯定会想：随机字符是什么，有多长，会产生什么奇怪的动作，应如何着手修复这个缺陷呢？

有效的软件缺陷描述如下所示：

① 简单，短小。只解释事实和演示、描述软件缺陷必需的细节，揭示错误的实质。说“一串随机字符”不是短小，应引用一些例子，特别是可以帮助开发人员找到原因的方式和线索；

② 单一。每个缺陷报告只针对一个软件缺陷；

③ 使用 IT 业界惯用的表达术语和表达方式；

④ 明确指明错误类型。

(3) 在报告软件缺陷时不做任何评价。由于测试人员的责任是找出软件中存在的缺陷，因此测试人员和开发人员之间很容易形成对立关系，在提交缺陷报告时要保持中立的态度，不要带有倾向性、个人观点和煽动性。例如：“这个问题在上一个版本中已经修复了，怎么这次又出现了呢？”“这个错误太低级了”等，这些语句都不应出现在缺陷报告中。

(4) 确保缺陷可以重现。如果开发人员不能重现提交的缺陷，将会影响开发人员的开发效率，也会影响测试人员自身的声誉。因此在提交缺陷之前一定要确保缺陷能够重现；对于严重程度较高的缺陷，一般要能保证按照预定步骤可以使其重复测试两次以上，对于随机产生的缺陷，要在其他机器上测试一下，看是否是自己机器的问题。

测试人员应该牢记上面这些关于报告软件缺陷的原则。这些原则几乎可以运用到任何交流活动中，尽管有时难以做到，但如果希望有效地报告软件缺陷，并使其得以修复，测试人员就必须遵循这些基本原则。

3. 缺陷报告的读者

缺陷报告的直接读者是软件开发人员和质量管理人员，来自市场和技术支持等部门的相关人员也可能需要了解缺陷情况。缺陷报告的读者最希望获得的信息包括：①软件缺陷报告中的缺陷；②报告的软件缺陷各自独立，缺陷信息更具体、准确；③缺陷的本质特征和复现步骤；④缺陷类型分布以及对市场和用户的影响程度。

4.3.2　缺陷报告撰写标准

软件缺陷报告又称软件问题报告，是软件工程技术规范中的一项重要内容，是软件测试过程中非常重要的文档。它记录了缺陷或问题发生的环境，如各种资源的配置情况、bug 的再现步骤以及 bug 性质的说明。更重要的是，它还记录着 bug 的处理过程和状态。bug 的处理过程从一定程度反映了测试的进程或被测软件的质量状况以及改善过程。

软件问题报告的编写或撰写是有要求和规范的，有相应的文档编写标准，如国际、军标及行业标准。图 4-13 给出了我国某行业的软件问题报告编制模板。

按照图 4-13 编制软件问题报告，需要事先填写软件问题报告单。该报告单的填写也有相应的要求、规范或标准。图 4-14 给出了某公司软件问题报告单的填写模板。

附录H 软件问题报告

H.1 编写目录
1 范围
1.1 标识
1.2 系统概述
1.3 文档概述
2 引用文件
3 术语和缩略语
4 软件问题报告
4.1 被评测软件的问题
4.1.1 测试发现的软件问题
4.1.1.1 小结
4.1.1.2 测试问题报告单
4.1.2 检查发现的问题
4.1.2.1 小结
4.1.2.2 检查问题报告单
4.2 其它问题

H.2 编写指南
1 范围
1.1 标识
写明本文档的：
a. 已批准的标识号；
b. 标题；
c. 本文档适用的被评测软件名称。
1.2 系统概述
概述本文档适用的系统和系统的用途。
1.3 文档概述
简要说明本文档的目的和用途。
2 引用文件
按文档号、标题、编写单位（或作者）和出版日期等，列出本文档引用的所有文件。
3 术语和缩略语
本章给出所有在本文档中出现的专用术语和缩略语的确切定义。
4 软件问题报告
4.1 被评测软件的问题
4.1.1 测试发现的软件问题
4.1.1.1 小结
对测试中发现的软件问题进行汇总。
4.1.1.2 测试问题报告单
按照《软件测试细则 ZZ-RW-03》的要求，填写"软件问题报告单"。
4.1.2 检查发现的问题
4.1.2.1 小结
对检查中发现的软件问题进行汇总。
4.1.2.2 检查问题报告单
按照《软件测试细则 ZZ-RW-03》的要求，填写"软件问题报告单"。
4.2 其它问题
描述评测发现的其它问题。按照《软件测试细则 ZZ-RW-03》的要求，填写"软件问题报告单"。

图 4-13　我国某行业的软件问题报告编制模板

软件问题报告单

编号：____________

系统名称及版本					
模块名称及版本					
模块版本信息	（文件创建日期，文件大小，相关其他文件信息等）				
报告人			联系电话		
发生日期			提交日期		
问题类型：	程序□	数据库□		文档□	其他□
要求完成日期			实际完成日期		
问题描述/影响（问题发现过程与曾经处理过程）：					
问题背景描述： 如：系统使用方，工程启动时间，工程预期结束时间，工程当前状态（测试、试运行还是已经投产），系统的版本号，软件的来源方式等。					
问题严重性：	□很严重：严重影响系统验收、投产或正常运行。 □严重：影响系统验收、投产或正常运行。 □一般：影响操作。				
问题影响到下一步工作描述：					
受理人		受理时间		承诺解决时间	
受理人意见：					
修改人			完成时间		
问题解决过程以及解决办法描述：					
问题解决结果并分析原因：					
报告人确认并签名：					

图 4-14　某公司软件问题报告单的填写模板

4.4　缺陷管理工具

缺陷管理是软件开发和软件质量管理的重要组成部分，是软件开发管理过程中与配置管理并驾齐驱的最基本管理需求。目前，随着人们对缺陷管理工具的需求逐渐增多且更加明确，国内外越来越多的公司进行相关管理工具的开发，包括缺陷管理工具的开发、提供高质量的商用工具。同时，人们渴望能够得到物美价廉的可用版本(当然大多数都有免费的试用版)。缺陷管理及缺陷管理工具的重要性和被人们所给予的重视程度越来越高。

缺陷管理工具用于集中管理软件测试过程中发现的错误，是添加、修改、排序、查寻、存储软件测试错误的数据库程序。

大型本地化软件测试项目一般测试周期较长，测试范围广，存在较多软件缺陷。如果对测试质量要求较高，并有支持多语言或本地化的要求，特别需要缺陷管理工具。

缺陷管理工具的使得查找和跟踪方便，对于大型本地化软件的测试，报告的错误总数可能成千上万个，如果没有缺陷管理工具，要查找某个错误，真是一件痛苦的事。

另外，缺陷管理工具的使用也使得跟踪和监控错误的处理过程和方法更加容易，既可以方便地检查处理方法是否正确，可以确定处理者的姓名和处理时间，作为工作质量的统计和考核的参考。

而且，缺陷管理工具的使用为集中管理提供了支持条件，为大大地提高管理效率提供了可能。例如，本地化服务商和软件供应商共享同一个错误跟踪系统数据库，各自负责处理己方需要处理的软件错误。对于需要对方提供更多信息的错误，可以通过改变错误的当前信息(状态、处理者、处理建议等)，使对方尽快处理。

最后，缺陷管理工具的使用使得整个缺陷管理的安全性得以提高，通过权限设置，不同权限的用户能执行不同的操作，保证只有适当的人员才能执行正确的处理；同时，能够保证缺陷处理顺序的正确性，根据当前错误的状态，决定当前错误的处理方法。最重要的是缺陷管理工具具有方便存储的特点，便于项目结束后的缺陷管理活动及历史过程存档，可以随时或在项目结束后存储，以备将来参考。

4.4.1　缺陷管理工具介绍

有些项目很简单，缺陷也就十几个、几十个，采用手工进行缺陷管理就可以了。而有些项目，特别是大型软件，会有成千上万个缺陷，甚至还有无法预计的缺陷。这时，手工进行缺陷管理就很不现实，而选用合理的缺陷管理工具便成为不可回避的问题。

本节主要介绍一些开源的缺陷管理工具，以便读者使用。例如 Mantis(免费)、Bugzilla (免费)、JIRA(免费)、TrackRecord(Compuware 公司)、ClearQuest(IBM Rational 公司)，这些都是专门的缺陷管理工具。此外，有些测试管理工具也具有缺陷管理的功能，如 HP 公司的 ALM、IBM Rational 公司的 TestManager，但这些都是商业软件。

商业软件有商业软件的好处，例如 MI 公司的 TestDirector，采用的是 B/S 构架模式、Windows 平台，可以定制流程、查询、功能域、用户角色及角色权限，可 e-mail 通知，可以生成各种报表并支持多种数据库，还可以与其他 MI 公司测试工具集成，安装配置也较为简单，有可优化的工作流，可使用 C 语言来改进优化系统。当然，开源软件也有自身的

优点：基本上都是基于 Web 的、提供源程序、免费使用、用户有更多的自由操作空间，等等。

基于 Web 的缺陷跟踪系统有很多好处，它简化了缺陷管理的相关工作：

(1) 实现地域上分散的项目人员高效协同工作，有效地降低软件测试成本，提高工作效率。

(2) 通过设置不同的用户权限，安全、准确地实现缺陷的管理和跟踪，且便于项目结束后的存档，以备将来参考。

(3) 系统维护简单，如果采用 B/S 结构，则只需要修改服务器端。现有的基于 Web 的缺陷跟踪系统所提供的功能都基本相同，但是在管理结构的实现上有一些不同，最简单的实现是一个系统有一个管理员，这个管理员负责管理所有的用户和项目。管理员的工作量会随用户和项目的增加而增加，通信和管理成本也会增加，因此这样的系统不适合大型项目或者地域分散的软件生产商采用。

下面简单介绍几个常用的开源缺陷管理工具。

1. Bugzilla

Bugzilla(免费，跨平台)是一款开源的 bug 追踪系统，可以用来帮助管理软件开发。Bugzilla 虽然是专门为 UNIX 定制开发的，但是在 Windows 平台下依然可以成功安装使用。

而且，Bugzilla 还能够被集成到 Testopia 系统(测试用例管理系统)中。Bugzilla 的强大功能表现在以下几个方面：

(1) 强大的检索功能。

(2) 用户可配置通过 e-mail 来公布 bug 变更。

(3) 历史变更记录。

(4) 通过跟踪和描述处理 bug。

(5) 附件管理。

(6) 完备的产品分类方案和细致的安全策略。

(7) 安全的审核机制。

(8) 强大的后端数据库支持。

(9) Web、XML、e-mail 和控制界面。

(10) 友好的网络用户界面。

(11) 丰富多样的配置设定。

(12) 版本间向下兼容。

2. BugOnline

BugOnline(开源)是一款开源的 bug 管理系统，功能强大，易于使用。BugOnline 基于 ASP.NET、SQL Server(包括 Express 版)及 AJAX 等技术。

BugOnline 具有如下一些特性：

(1) 在线消息及 e-mail 自动通知功能。当有新 bug 及 bug 分配给用户时，会自动通知用户。

(2) 优秀的人员分配、工作量统计功能。

(3) 基于项目角色的权限管理、工作规划及流程化。
(4) bug 状态统计，便于掌控项目进度。
(5) 基于 SSL 的数据传输，确保数据不被截取，保证安全性(也可设定为非 SSL)。
(6) 强大的报表功能。

3. BugZero

BugZero(免费，开源，跨平台)是一款多功能、基于网络并在浏览器下运行的 bug 缺陷管理和跟踪系统，可用来记录、跟踪并归类处理软件开发过程中出现的 bug 和硬件系统中存在的缺陷。BugZero 还是一款完整的服务管理软件，集成了服务台热线流程管理，可用来记录各种日常事务、变更请求和问题报告，并追踪和处理各种客户询问、反馈和意见。

BugZero 提供了一个可靠的中央数据库，使得公司内部团队成员以及外部客户能在任何地点、任何时间进行协调和信息交流，并且使任何记录都有据可查。它使用户省时省力。BugZero 不但使用方便，而且功能齐全，变通性好，能够灵活设置各种实际工作流程，满足特定业务和产品环境下的需求。这种灵活、易用的缺陷跟踪流程不仅增强了项目开发的质量，同时也提高了整个机构的生产效率。

4. 其他开源缺陷管理工具

BugTracker 是一个完整的 bug/issue 管理系统，以 Java Servlet 作为 Web 前台，以 MySQL 数据库作为后台。

BugFree 是借鉴微软的研发流程和 bug 管理理念，使用 PHP+MySQL 独立编写的一个 bug 管理系统。它简单实用、免费并且开放源代码(遵循 GNU GPL)。

JTrac 是一个开源且可高度配置的用于缺陷追踪的 Web 应用程序。它可以跟踪网络应用程序，可方便地实现定制，增加了自定义字段。其特点包括可定制的工作流程、实地许可、电子邮件集成、文件附件和详细历史记录查询。

BugNet 是一个不错的开源 bug 跟踪和项目管理系统。

eTraxis 是基于网页的免费 bug 跟踪系统。其主要特点是完全自定义模板、先进的过滤器、LDAP 支持、电子邮件通知、订阅报刊、提醒、灵活的权限管理、图形化的项目指标等。

4.4.2　缺陷管理工具 Mantis 及其应用

Mantis 同样是一款开源的软件缺陷管理工具，是一个基于 PHP 技术的轻量级缺陷跟踪系统，其功能与商用的 JIRA 系统类似，都以 Web 操作的形式来提供项目管理及缺陷跟踪服务。Mantis 在功能上可能没有 JIRA 那么专业，界面也没有 JIRA 漂亮，但在实用性上足以满足中小型项目的缺陷管理及跟踪需求。Mantis 包括客户端浏览器、Web 服务器和数据库服务器。当然，Web 服务器和数据库服务器也可以是同一台主机。重要的是它是开源的，不需要付任何费用。不过 Mantis 目前的版本还存在一些问题，期待在今后的版本中能够得以完善。

1. Mantis 功能介绍

Mantis 基于 PHP+MySQL，可以运行于 Windows/UNIX 平台上。作为一个 bug 管理系

统，其适用性是否符合实际工作的需要是至关重要的。Mantis 基本可以满足 bug 管理日常流程。而且，Mantis 是 B/S 架构的 Web 系统，如果今后有需要，还可以配置到 Internet 上，实现异地 bug 管理。在 Mantis 系统中，有如下几种角色：管理员、经理、开发人员、修改人员、报告人员、查看人员。每个角色所拥有的权限是不一样的，从大到小依次排列是：管理员→经理→开发人员→修改人员→报告人员→查看人员。

Mantis 的特点是免费、简洁灵活，B/S 结构的 Web 系统比较适合分布式协作开发和测试。关于 Mantis 的详细信息和技术支持，可访问 http://www.mantisbt.net/。

1) Mantis 的基本特征

(1) 个人可定制的 e-mail 通知功能，每个用户可根据自身的工作特点只订阅相关的缺陷状态邮件。

(2) 支持多项目、多语言。

(3) 权限设置灵活，不同角色有不同权限，每个项目可设为公开或私有状态，每个缺陷也可设为公开或私有状态，每个缺陷可以在不同项目间移动。

(4) 主页可发布项目相关新闻，方便信息传播。

(5) 方便的缺陷关联功能。除重复缺陷外，每个缺陷都可以链接到其他相关缺陷。

(6) 缺陷报告可打印或输出为 CSV 格式。支持可定制的报表输出，可定制用户输入域。

(7) 有各种缺陷趋势图和柱状图，为项目状态分析提供依据，如果不满足要求，可以把数据输出到 Excel 中进一步分析。

(8) 流程定制不够方便，但流程可满足一般的缺陷跟踪。

(9) 可以实现与 CVS 的集成，即实现缺陷和 CVS 仓库中的文件相关联。

(10) 可以对历史缺陷进行检索。

2) Mantis 系统中缺陷状态的转换

缺陷状态是描述软件缺陷处理过程所处阶段的一个重要属性。对应于不同的状态，软件测试人员能确定对该问题的处理已经发展到什么阶段，还需要进行哪些工作，需要哪些人员的参与等信息。缺陷跟踪系统的状态比较复杂，这也是缺陷管理中的难点。在缺陷跟踪管理过程中，将缺陷记录划分为不同的阶段、不同的状态来进行标记。Mantis 系统将缺陷的处理状态分为 New、Active、Invalid、Later、Resolve、Reopen、Closed 七种，如图 4-15 所示。

(1) 一个新的缺陷被提交，即为 New。

(2) Active，刚提交的缺陷，在项目经理确认并分发给研发人员修改前所处的状态。

(3) Invalid，已提交的缺陷在当前版本中已不是问题或不需要修改。

(4) Later，提交的缺陷在当前研发阶段无法对其进行修改。

(5) Resolve，经软件工程师修改或给出相关意见后，等待测试人员验证时所处的状态。

(6) Reopen，已经关闭的缺陷重新出现，测试人员将其状态设置为 Reopen，分发缺陷时的操作与状态为 New 时的类似。

(7) Closed，最终修改正确或不正确的缺陷报告，经过验证或项目经理同意后，可以关闭。处于关闭状态的缺陷报告可表现为已改正、符合设计、不能重现、不能改正、由报告人撤回。

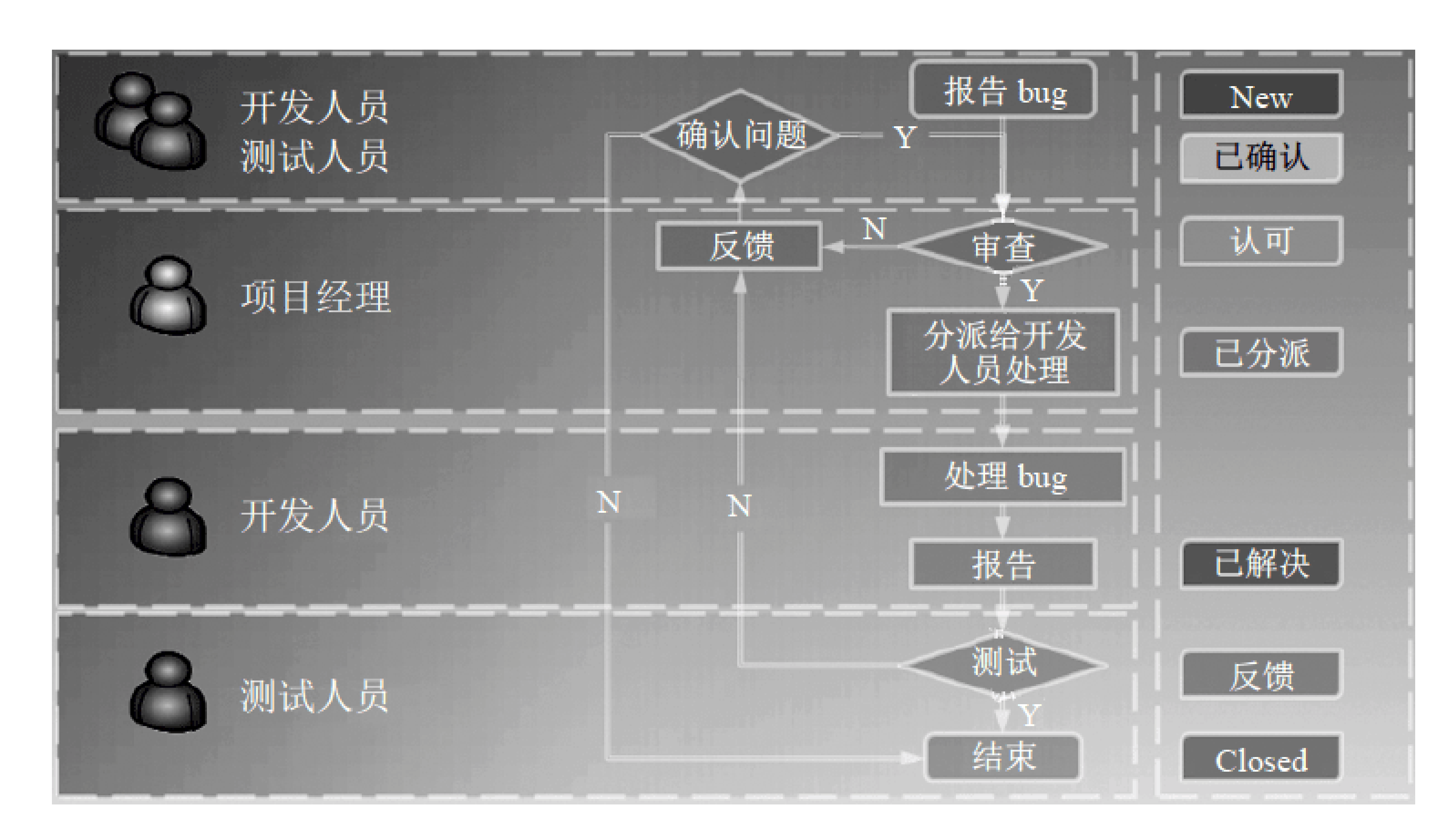


图 4-15　Mantis 缺陷状态转换图

3) Mantis 用户角色及权限的管理

在一个测试项目中，存在各种不同的身份，比如项目经理、测试经理、开发经理、程序员、测试员等。不同身份的用户使用系统时可以执行的操作理应是不同的，例如不能让一个测试员来进行用户分工的工作。另一方面，权限的要求是以对象为中心的，比如对于缺陷这个对象，它的填报信息只能由填报该缺陷的测试员来修改和维护，其他任何人都不能具有同等的操作权限。

Mantis 中用户角色、登录权限及方式如表 4-7 所示。

表 4-7　Mantis 中用户角色、登录权限及方式

用户	权限	工作范围
管理员(administrator)	高	管理和维护整个系统
项目经理(manager)	↓	对整个项目进行管理
开发人员(developer)	↓	负责整个软件的开发
修改人员(updater)	↓	负责修改 issue
报告人员(reporter)	↓	负责提交 bug 报告
查看人员(viewer)	低	查看 bug 流程及情况

4) Mantis 的软件缺陷属性的定义

软件缺陷是按照能准确发现缺陷目标进行分类的，分类之间应无重叠，分类体系应覆盖所有的缺陷类型；要与软件生命周期相结合。传统的分类方法可分为按照缺陷的来源和缺陷错误性质这两种，如 Putnam 等人提出的分类方法和正交缺陷分类法以及 IEEE 制定的软件异常分类标准等。正交缺陷分类法定义的软件缺陷的 13 个属性在 Mantis 中得到了实现。

(1) 缺陷编号：缺陷的唯一标识。

(2) 模块信息：缺陷涉及的模块信息，包括模块名称、缺陷处理负责人、模块版本。

(3) 测试版本：描述的是该缺陷发现的测试版本号。

(4) 对应用例编号：发现该缺陷时运行的测试用例编号，通过该编号可以建立起测试用例和缺陷之间的联系。

(5) 缺陷状态：缺陷的即时状态，如 New、Active、Invalid、Later、Resolve、Reopen、Closed 等。

(6) 持有人：描述缺陷当前由谁负责，如果这个持有人是程序员，那么这个缺陷正在被修改；而如果持有人是测试员，那么这个缺陷正在等待被确证。

(7) 报告者：报告缺陷的测试人员的编号或用户名。

(8) 报告日期：缺陷填报的日期。

(9) 重现性：可重现或不可重现。

(10) 重现步骤：和测试用例相关，描述的是发现这个缺陷的步骤。

(11) 严重等级：可定制，默认为 4 级——Pl(致命)、P2(严重)、P3(一般)、P4(轻微)。

(12) 缺陷类型：可定制，默认分为功能缺陷、用户界面缺陷、边界值相关缺陷、初始化缺陷、计算缺陷、内存相关缺陷、硬件相关缺陷、文档缺陷。

(13) 缺陷优先级(报告者)：可定制，默认分为必须修复、立即修复、应该修复、考虑修复。

2. Mantis 的具体功能介绍

1) 多项目管理

在系统页面上单击 Manage | Manage Projects，可以进入项目管理界面。上面显示了已创建的项目列表，单击 Create New Project，可进入新建项目页面。可以设定新项目当前的状态，项目状态有 development、release、stable 和 obsolete 这几种。在已建项目列表中可以修改项目数据，包括修改项目状态(将项目修改为公开或私有)，添加和修改子项目，为该项目添加和修改 Categories，添加和修改项目发布版本，定义项目可使用的用户自定义域，添加和修改项目用户及其权限属性。

2) 问题录入

在系统页面上单击 Report Issue，可进入问题录入界面。如果在单击前，右上角项目选择为 All Project，那么在填报问题前需要先选择要填报的项目。可以选中 Make Default，这样在每次填报进入该界面时，所选择的就是默认项目了。在问题填报界面选择并输入 Category、Reproductbility、Impact、Severity、Summary、Description、Additional Information 等信息，单击 Submit Report 按钮即可。在问题录入界面中还可以添加和上传附件。

3) 问题查询和关键词检索

在系统页面上单击 View Issues，可进入问题查询结果界面。在项目选择中，可以选择项目，查看所属项目问题，单击查询结果区的字段名称，可以进行排序显示。界面上方区域是问题检索条件区，可以一览当前查询结果的查询条件，也可以单击每个查询条件以修改该查询条件选项。修改各查询条件参数，单击 Apply Filter 按钮即可。在该查询界面上，每个查询条件只能定义单一值。如果需要定义多值查询，可以在查询结果界面上单击 Advanced Filters，界面刷新后，单击某查询条件，便可以选择多个选项进行查询。在查询结果界面的查询条件区，可以在 search 文本框中输入所要查询问题信息中的关键词，单击 Apply Filter 按钮，即可显示含有该关键词的所有历史问题。可以将当前查询条件保存为过滤器，以便快速选择得到查询结果。在查询区单击 Save Current Filter，可以命名并保存当前过滤器。若当

前过滤器的查询条件与已有过滤器的相同，那么保存界面时会提示“This particular query appears to already exist”。输入待保存的过滤器，保存即可。在查询界面上单击 Manage Filters，可以管理过滤器。

4) 问题更新

(1) 单击 Assign to 按钮，将问题安排给相关人员解决。

(2) 可以单击 Due to 按钮，添加问题责任人。

(3) 单击 Change Status to，修改问题状态。

(4) 单击 Monitor Issue，可以跟踪该问题。

(5) 单击 Create Clone，可以克隆一个新问题。

(6) 单击 Move Issue，可以将问题在不同项目间进行移动。

(7) 单击 Delete Issue，可以删除该问题。

(8) 也可以单击 My View 或查询结果界面上某条问题前的图标，进入问题详细界面。单击按钮可以直接下载问题的附件。也可以在系统菜单右侧输入问题编号，即可进入问题详细界面。单击 Update Issue，可以修改问题的属性数据。

5) 问题讨论

在问题详细界面的后面添加 Note 信息，以便将该问题的讨论、交互信息记录下来。讨论信息可以进行编辑、删除，也可以被修改为私有状态。

6) 问题关联关系

在问题详细界面，可以设置该问题与其他问题之间的关联关系。每个问题都可以链接到其他相关问题。链接的关系分为 related to、parent of、child of、duplicate、has duplicate 几种。可以对当前链接的问题进行删除，有关系冲突的可以设置最新的关联关系。对于存在父子关系的问题，如果子问题没有解决，在父问题的关联关系中会显示 Not all the children of this issue are yet resolved or closed，以提示子问题没有被全部解决。

对于子问题没有全部解决的父问题，如果要将其状态设置为解决或关闭，则会在设置状态界面的上方提示“ATTENTION. Not all the children of this issue are yet resolved or closed. Before resolving/closing a parent issue，all the issues related as child with this one should be resolved or closed”。通过单击问题详细界面上 Relationships 区域中的 Relation Graph，可以查看该问题的关联关系图。单击 Dependency Graph，可以查看当前问题的依存关系图。在关联关系图和依存关系图中，当光标移动到各问题的 ID 方框时，会显示该问题 ID 的 Status 和 Summary。

7) 集成 CVS

当将 CVS 文档提交给 CVS 服务器时，在 log message 中添加 issue #nnnn，提交后，即可将该提交信息插入到 issue #nnnn 的 Note 文本框中。单击该提交的文件版本链接，弹出 commit 前后版本比较信息界面。通过单击系统菜单 Docs | CVS Web，可以浏览 CVS 仓库。

8) 个人显示和 e-mail 通知设定

个人可定制的 e-mail 通知功能，使得每个用户可根据自身的工作特点而只订阅相关的缺陷状态邮件。在系统菜单中单击 My Account，进入用户个人设定界面。可以在 My Account 选项中修改用户密码和用户邮件地址，在 Preferences 中设定默认设置，可以对不同问题状

态设定是否接收 e-mail。还可以设定自己的系统界面语言，为了实现多语言使用，一般使用各对应语言的 UTF-8 选项，可供选择的有：

(1) english_utf8。

(2) chinese_simplified_utf8。

(3) chinese_tranditional_utf8。

(4) japanese_utf8。

在 Profiles 中可以设定 Platform、Operating System、Version 等。

9) 统计分析、报表生成和输出

在系统菜单中单击 Summary，以显示该项目下问题统计 Synthesis 情况，包括按 Project、Status、Date、Resolution、Severity、Category 等进行统计的结果。单击 Summary 表上方的图表按钮，分别有 Per state、Per severity、Per impact、Per category 和 Per resolution 的统计表。后面仅列出了 Per state 的表截图。单击 Advanced Summary，可以显示总体统计图表，包括 Cumulative By Date 图。通过后台系统文件的设定，可以添加和修改统计图表。单击 Print Report，可打印当前项目下的问题。可以选择性地将问题导出至 Excel 或 Word 文件中，也可通过预览功能在 IE 中显示，并可另存为 HTML 文件。对于问题导出，还可以在问题查询结果界面中，通过单击 CSV Export，将问题导出为 CSV 文档。在问题查询结果界面中单击 Print Report，可以进入打印报告界面。

10) 用户管理

使用管理员账户进入系统，单击系统菜单 Manage | Manage Users，进入用户一览界面。可以按用户 ID 的字母顺序筛选用户。可以单击各用户以修改其权限和信息，也可以单击 Prune Accounts 来阻止未登录的用户。单击 Create New Account 建立新账户时，可以选择是否激活该账户，也可以设定用户权限。用户权限包括 viewer、reporter、updater、developer、manager 和 administrator(角色可以定制)。权限可以在系统权限设置中进行控制。

11) 自定义域

通过单击系统菜单 Manage | Manage Custom Fields，用户可以自行添加和修改自定义域，添加数量没有限制。自定义域的类型有 String、Numeric、Float、Enumeration、Email、Checkbox、List、Multiselection List、Date 等。可以设置是否在报告、更新、解决、关闭界面中显示和必填，以及是否仅在高级查询条件界面中显示。

12) 系统设置

使用管理员权限进入系统，单击 Manage | Manage Configuration，进入系统设置界面。Permissions Report 界面显示了当前系统的权限分配情况。在 Workflow Thresholds 界面，可以设置不同角色权限。在 Workflow Transitions 界面，可以设置工作流。可以根据公司流程进行定制。可以设定问题各状态的最低权限角色。

13) 新闻发布

新闻发布后，可以在系统菜单 Main 中进行显示，这样用户一进入系统就可以看到。

3. Mantis 应用环境的建立

要安装、运行 Mantis，有两种主流的环境配置可供选择：IIS+PHP+MySQL+Mantis 和

Apache+PHP+MySQL+Mantis，下面介绍后一种。由于单个配置相当复杂，且容易出错，这里通过安装 XAMPP 配置环境。

XAMPP(Apache+MySQL+PHP+Perl)是一个功能强大的软件集成包。这个软件包原来的名字是 LAMPP，但是为了避免误解，最新的几个版本就改名为 XAMPP。它可以在 Windows、Linux、Solaris、Mac OS X 等多种操作系统中安装使用，支持多语言：英文、简体中文、繁体中文、韩文、俄文、日文等。许多人通过他们自己的经验认识到安装 Apache 服务器不是件容易的事。如果想添加 MySQL、PHP 和 Perl，那就更难了。XAMPP 是一个易于安装且包含 MySQL、PHP 和 Perl 的 Apache 发行版。

1) 安装 XAMPP

这个安装比较简单，直接安装就行，这里安装的是 XAMPP Windows 3.2.2 版本。下载网址为 http://www.apachefriends.org/zh_cn/xampp.html。

安装完之后，打开控制面板，表明 Apache 和 MySQL 正在运行，单击图 4-16 中 Apache 一行中的 Admin 按钮，弹出 XAMPP 页面，选择中文后，单击左边的“安全”，出现 XAMPP 安全页面，如图 4-17 所示。

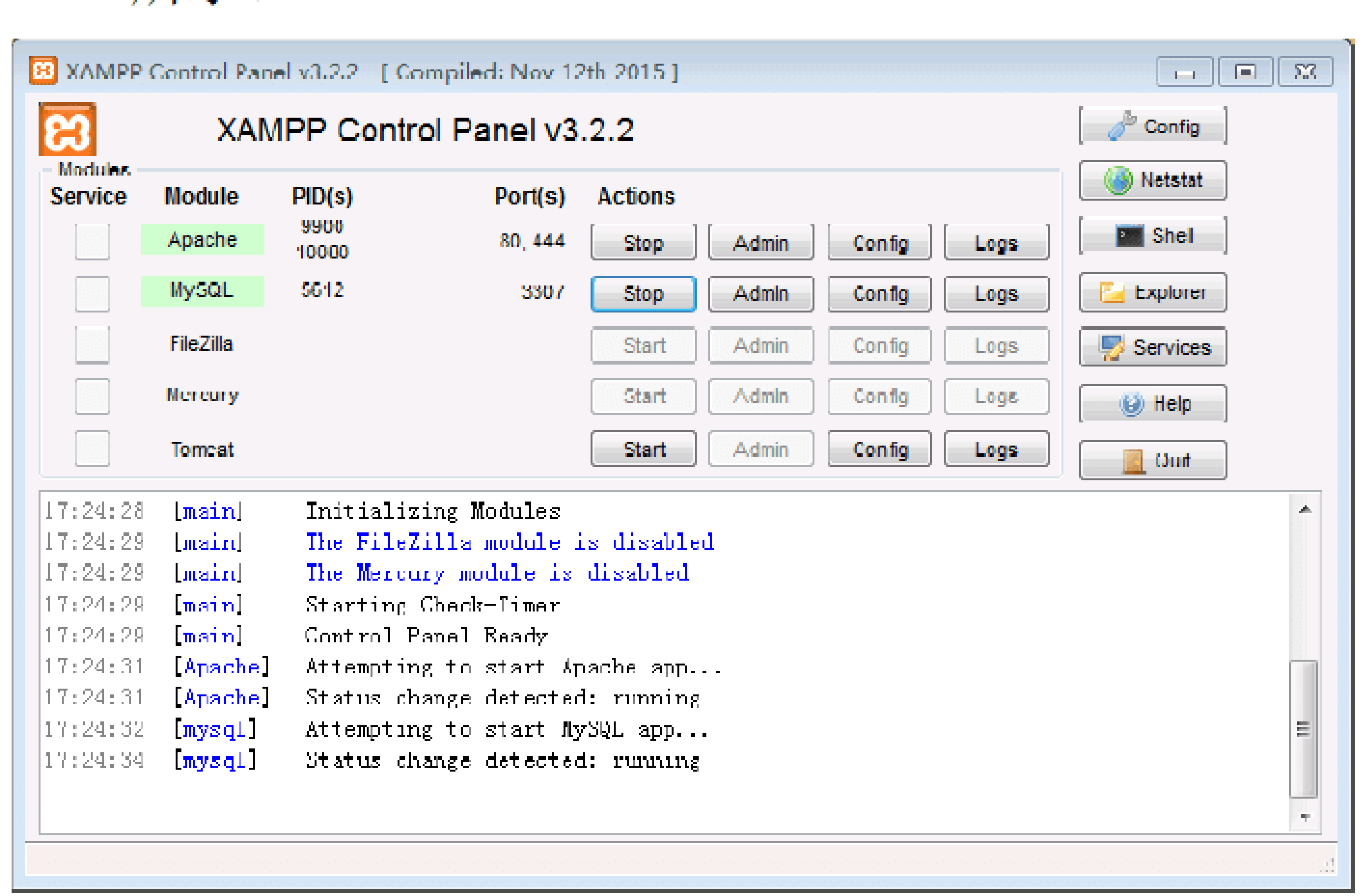


图 4-16　XAMPP 控制面板



图 4-17　XAMPP 安全页面

修改 MySQL 中的 root 密码为“root”，登录 http://127.0.0.1/，打开 phpMyAdmin 页面进行修改，如图 4-18 所示。密码修改成功。

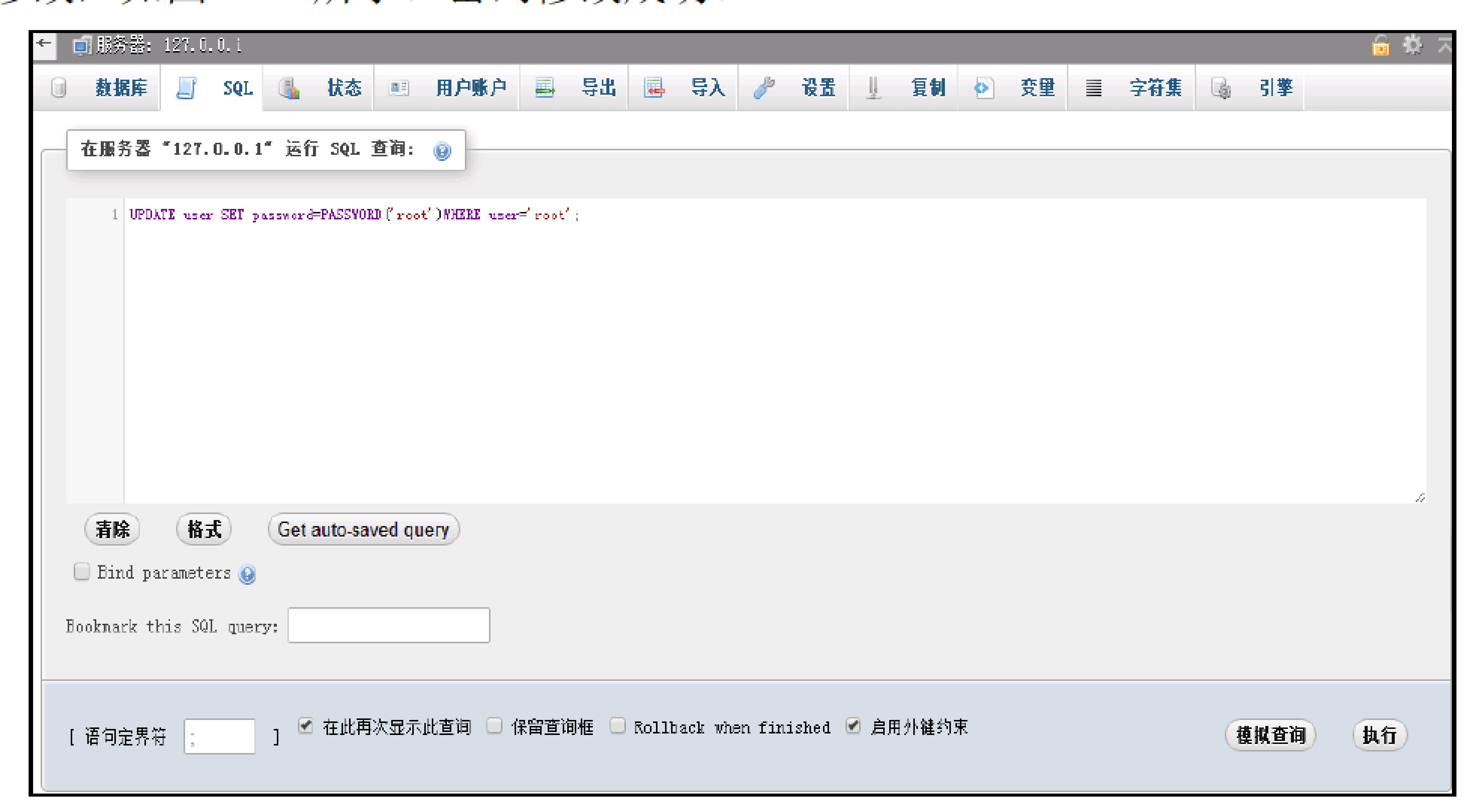


图 4-18　密码修改成功

2) 安装 Mantis

这里选择安装最新的 Mantis 发布版本 1.2.19，登录 http://localhost/phpmyadmin，用户名和密码均为 root，创建数据库 mantis，如图 4-19 所示。

把 mantis 1.2.19 解压到 E:\xampp\htdocs\目录下，并改名为 mantis。

打开 IE，输入 http://localhost/mantis，即可进入安装界面，如图 4-20 所示。

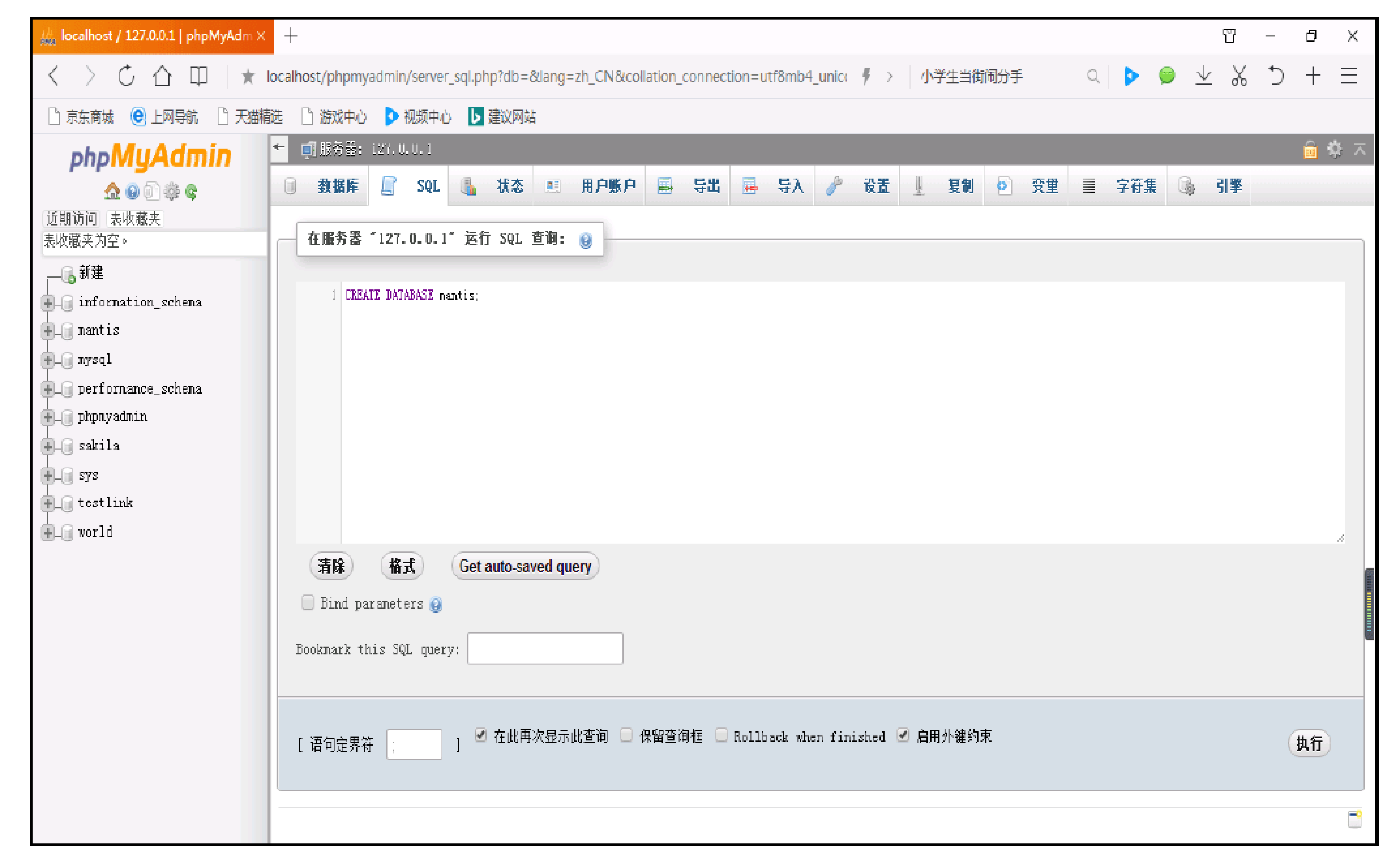


图 4-19　创建数据库 mantis

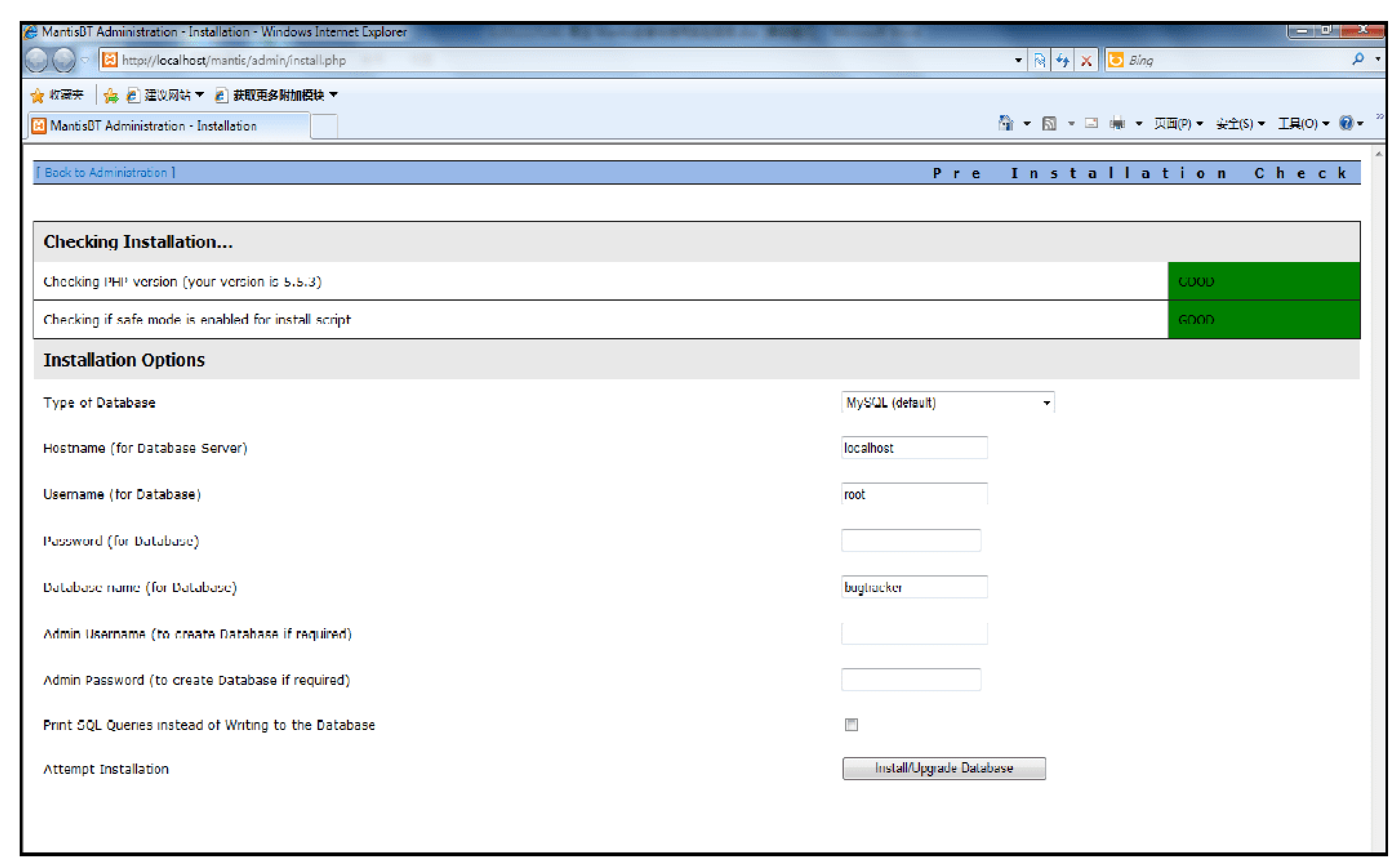


图 4-20　Mantis 安装界面

(1) 在安装界面，除了 Database name 填 mantis 外，上下两个 user name 和 password 都填 root。如果安装成功，后面的状态栏为全绿，如图 4-21 所示。

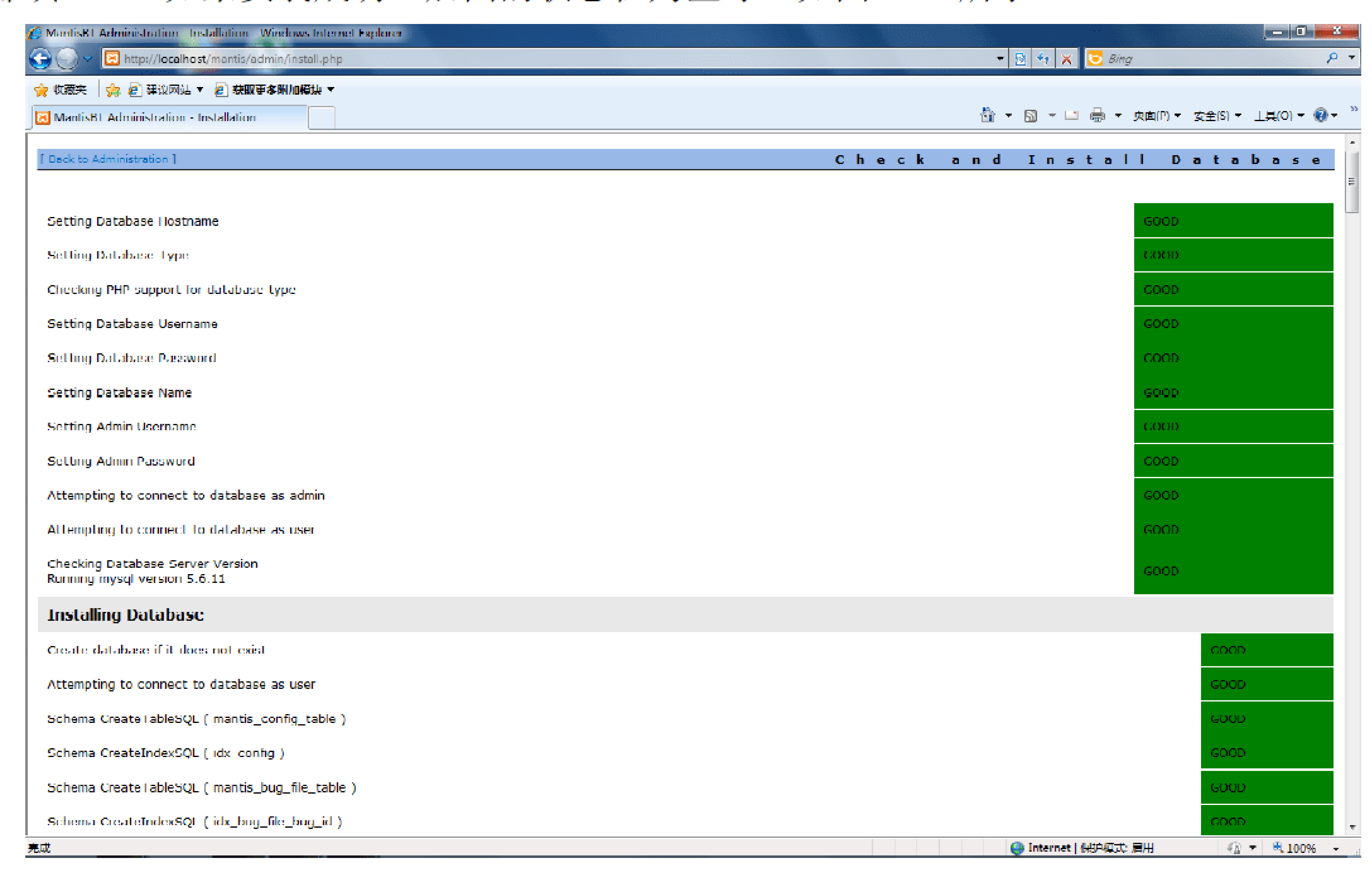


图 4-21　Mantis 安装成功

(2) 打开 IE，输入 http://localhost/mantis，可以看到 Mantis 登录页面，如图 4-22 所示。初始用户可以使用默认用户名 administrator 和密码 root 登录进去，进行管理设置。

图 4-22　Mantis 登录界面

3) Mantis 的其他设置

Mantis 的设置是这样保存的：在 config_defaults_inc.php 中保留 Mantis 的默认设置，用户自己的设置信息保存在 config_inc.php 中，如果某个选项在 config_inc.php 中已设置，则系统使用 config_inc.php 中的设置，否则使用 config_defaults_inc.php 中的系统默认设置。

config_inc.php.sample 是 Mantis 给出的一个用户设置文件示例。sample 中给出的一些设置是一定需要修改的，比如 MySQL 数据库的连接参数、管理员的邮箱；其他的则要根据实际情况进行修改。config_inc.php 文件中的设置很简单，各个参数的意义参见 config_defaults_inc.php，其对每个参数都有详细的解释。

```
$g_use_iis = ON;                                    # 使用 IIS
$g_show_version = OFF;                              # 不在页面底部显示 Mantis 的版本号
$g_default_language = 'chinese_simplified';         # 默认语言为简体中文
$g_show_project_menu_bar = ON;                      # 显示项目选择栏
$g_show_queries_count = OFF;                        # 在页脚处不显示执行的查询次数
$g_default_new_account_access_level = DEVELOPER;    # 默认用户级别
$g_use_jpgraph = ON;                                # 使用图形报表
$g_jpgraph_path = 'C:/PHP/includes/JPGraph/src/';   # JPGraph 路径
$g_window_title = 'Mantis Bug 跟踪管理系统';         # 浏览器标题
$g_page_title = 'Mantis Bug 跟踪管理系统';           # 页面标题栏
$g_enable_email_notification = ON;                  # 开通邮件通知
$g_smtp_host = 'smtp.mail.net';                     # SMTP 服务器
$g_smtp_username = 'mailuser';                      # 邮箱登录用户名
$g_smtp_password = 'mailpwd';                       # 邮箱登录密码
$g_use_phpMailer = ON;                              # 使用 PHPMailer 发送邮件
$g_phpMailer_path = 'C:/PHP/includes/PHPMailer/';   # PHPMailer 的存放路径
$g_phpMailer_method = 2;                            # PHPMailer 以 SMTP 方式发送 e-mail
$g_file_upload_ftp_server = 'ftp.yourftp.com';      # 上传文件 FTP
$g_file_upload_ftp_user = 'ftpuser';                # FTP 登录用户名
$g_file_upload_ftp_pass = 'ftppwd';                 # FTP 登录密码
$g_short_date_format = 'Y-m-d';                     # 短日期格式，Y 大写表示以 4 位表示年数
$g_normal_date_format = 'Y-m-d H:i';                # 普通日期格式
$g_complete_date_format = 'Y-m-d H:i:s';            # 完整日期格式
```

设置 Mantis 邮件服务

首先修改 E:\xampp\php 目录下的 php.ini 文件，查找 SMTP，将 SMTP = localhost 改为发件服务器(这里以 163 邮箱为例)，如 SMTP = smtp.163.com；查找 sendmail_from，并修改为 sendmail_from = xxx@163.com(设置邮箱地址全称)。

然后在 E:\xampp\htdocs\mantis 目录下的 config_inc.php 文件中，添加如下代码：

```
$g_phpMailer_method = 2;
$g_smtp_host='smtp.163.com';
$g_smtp_username='xxx';                    //邮箱用户名
$g_smtp_password='yyy';                    //邮箱密码
```

最后在同一目录下的 config_default_inc.php 文件中做如下修改：

```
$g_phpMailer_method = PHPMAILER_METHOD_SMTP;
$g_smtp_host='smtp.163.com';
$g_administrator_email='xxx@163.com';      //邮箱地址全称
$g_webmaster_email='xxx@163.com';
$g_from_email='xxx@163.com';
$g_return_path_email='xxx@163.com';
```

设置图形报表

默认情况下，Mantis 的图形报表是关闭的，需要安装图形报表模块才能打开图形报表。

下载 JpGraph，从 http://jpgraph.net/download 下载 JpGraph 的安装文件，目前最高版本是 3.5.0b1。

将 jpgraph-3.5.0b1.tar.gz 解压缩，把其中的子目录 src 复制到 mantis\library 目录下，并改名为 jpgraph。

修改 E:\xampp\php 目录下的 php.ini 文件，将“;extension=php_gd2.dll”前面的分号删除，确保 PHP 有加载 JpGraph 使用的动态库。

安装插件：登录 Mantis(管理员权限)，执行“管理”→“插件管理”→“安装 Mantis Graph 1.0 插件”，如图 4-23 所示。



图 4-23　安装 Mantis Graph 插件

修改Mantis Graph插件配置，将要使用的图形库修改为JpGraph，将JpGraph库系统路径设置为对应的JpGraph路径，例如E:\xampp\htdocs\mantis\library\jpgraph，如图4-24所示。



图 4-24　Mantis Graph 插件配置

单击 Update Configuration，现在再打开 Mantis 的统计页面，可以看到多了分别按状态等进行统计的图形报表，包括柱形图、饼图和线图。

如果使用的界面语言是简体中文，那么将会看到图形中的汉字都是乱码，这是由于 Mantis 对 JpGraph 的编码设置不正确造成的。JpGraph 会自动将汉字转换为 UTF-8 编码，但是需要在调用 JpGraph 的时候为标题等设置字体，Mantis 没有做这个操作，因此汉字显示出来都是乱码。解决方法如下：

(1) 修改 mantis\library\jpgraph\jpgraph_ttf.inc.php：

```
elseif( $aFF === FF_SIMSUN ) {
        // Do Chinese conversion
        if( $this->g2312 == null ) {
            include_once 'jpgraph_gb2312.php' ;
            $this->g2312 = new GB2312toUTF8();
        }
        return $this->g2312->gb2utf8($aTxt);
    }
```

将其改为：

```
/*elseif( $aFF === FF_SIMSUN ) {
        // Do Chinese conversion
        if( $this->g2312 == null ) {
            include_once 'jpgraph_gb2312.php' ;
            $this->g2312 = new GB2312toUTF8();
        }
        return $this->g2312->gb2utf8($aTxt);
  }*/
elseif( $aFF === FF_SIMSUN ) {
        return $aTxt;
}
```

(2) 修改 mantis\plugins\Mantisgrpah\pages\config.php。

a) 增加字体 simsun。

将如下代码：

```
$t_current_font_selected = array('arial' => false,
```

改为：

```
$t_current_font_selected = array('simsun' => false, 'arial' => false,
```

b) 配置页面将显示新增加的 simsun 字体(宋体)。

将如下代码：

```
<label><input type="radio" name="font" value="arial"<?php echo print_font_checked( 'arial' )?>/>
Arial</label><br />
```

改为：

```
<label><input type="radio" name="font" value="simsun"<?php echo print_font_checked( 'simsun' )?>/>宋
体</label><br />
<label><input type="radio" name="font" value="arial"<?php echo print_font_checked( 'arial' )?>/>
Arial</label><br />
```

注意：

因为直接使用中文，为了不显示为乱码，需要把该代码文件转换成 UTF-8 编码格式，在文件另存为时选择即可。

(3) 修改 mantis\plugins\MantisGraph\pages\config_edit.php。

将如下代码：

```
if ( plugin_config_get( 'font' ) != $f_font ) {
    switch ( $f_font ) {
        case 'arial':
```

改为：

```
if ( plugin_config_get( 'font' ) != $f_font ) {
    switch ( $f_font ) {
        case 'simsun':
         case 'arial':
```

(4) 修改 mantis\plugins\MantisGraph\core\graph_api.php。

将如下代码：

```
$t_font_map = array('arial' => FF_ARIAL,
```

改为：

```
$t_font_map = array('simsun' => FF_SIMSUN, 'arial' => FF_ARIAL,
```

(5) 修改 MantisGraph 插件配置，执行“管理”→“插件管理”→“Mantis 图表 1.0”，编辑配置，修改字体为“宋体”，如图 4-25 所示。

这样，图形报表就可以显示中文了。

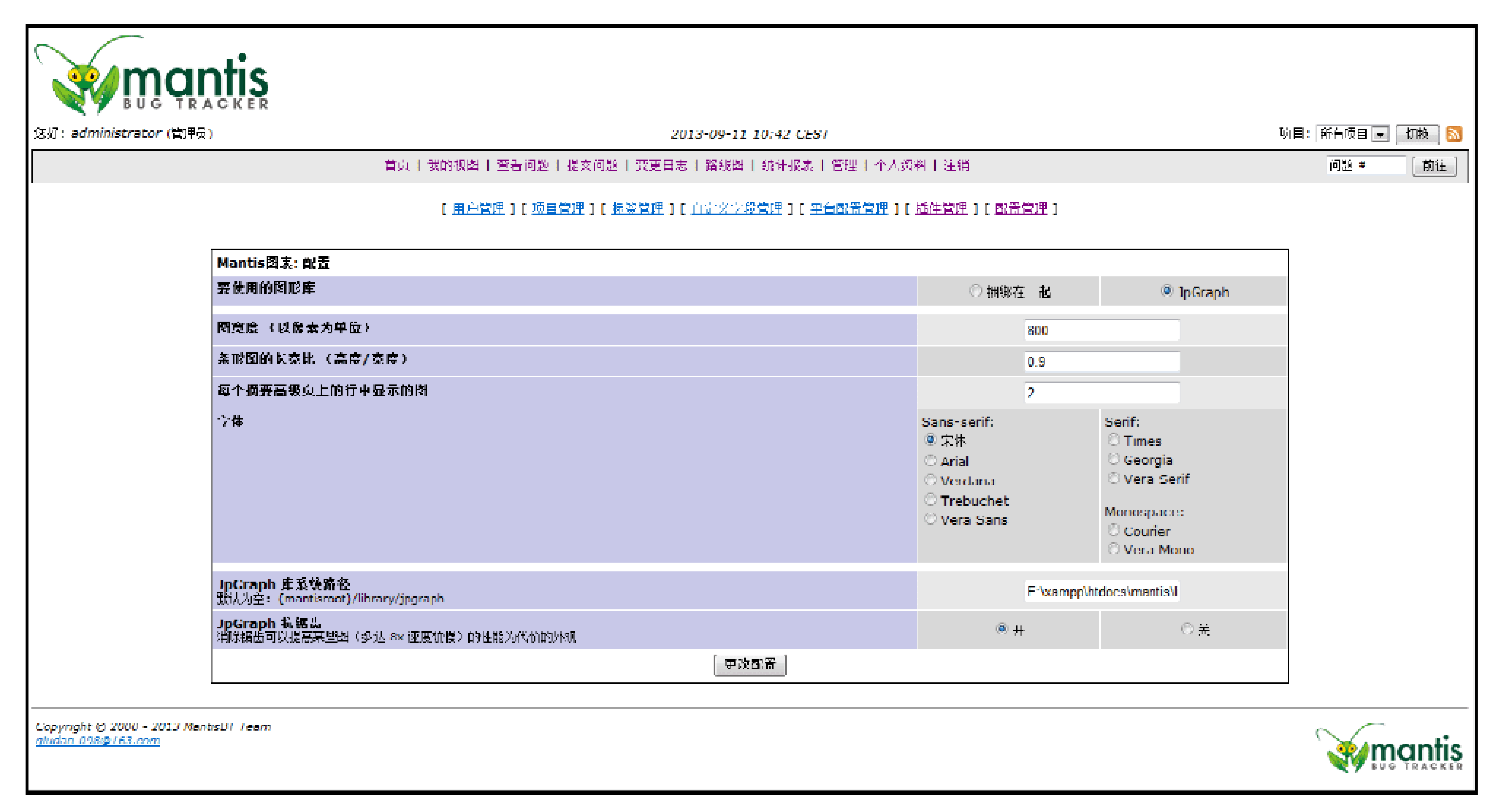


图 4-25　修改字体为“宋体”

3. Mantis 应用流程

在 Mantis 系统中，分别有如下几个角色：管理员、经理、开发人员、修改人员、报告人员、查看人员。每个角色所拥有的权限是不一样的，权限从大到小依次排列是：管理员→经理→开发人员→修改人员→报告人员→查看人员。

首先来看主页，在主页中可以看到以下信息：

(1) 指派给登录用户且还未解决的 Issue 数目，单击该数目链接，就可以进入 Issue 列表，直接对这些 Issue 进行操作。

(2) 由登录用户报告还未解决的 Issue 数目。

(3) 最后一次登录该系统的时间。

切换项目：单击界面右上角的下拉式菜单来切换所选项目。

跳转到该 Issue 编号：根据 Issue 编号可以进行查询，直接进入该 Issue 的详细信息界面，进行相应操作。

转向其他操作界面：单击主页面上方的菜单栏，即可进入相应的操作界面，如图 4-26 所示。



图 4-26　Mantis 的有关操作界面

下面将详细说明各个角色在系统中的具体权限和工作职责。

1) 管理员

管理员是管理整个系统运作的工作人员，他不仅是整个系统操作流程中权限最高的工作人员，而且还可以对项目和用户账户进行创建和管理等，下面将详细说明。管理员登录系统之后，可以先进入自己的主界面，然后再根据工作要求，选择界面上方的菜单栏来进入相应的界面。

我的视图

我的视图(My View)界面如图 4-27 所示。

bug 根据其工作状态被分类成几个表格来显示，符合这些工作状态的 bug 都被一一罗列出来。

① 分派给我的(未解决)，assigned。

② 未分派的，unassigned。

③ 我报告的，reported by me。

④ 已解决的，resolved。

⑤ 最近修改，recently modified。

⑥ 我监视的，monitored by me。

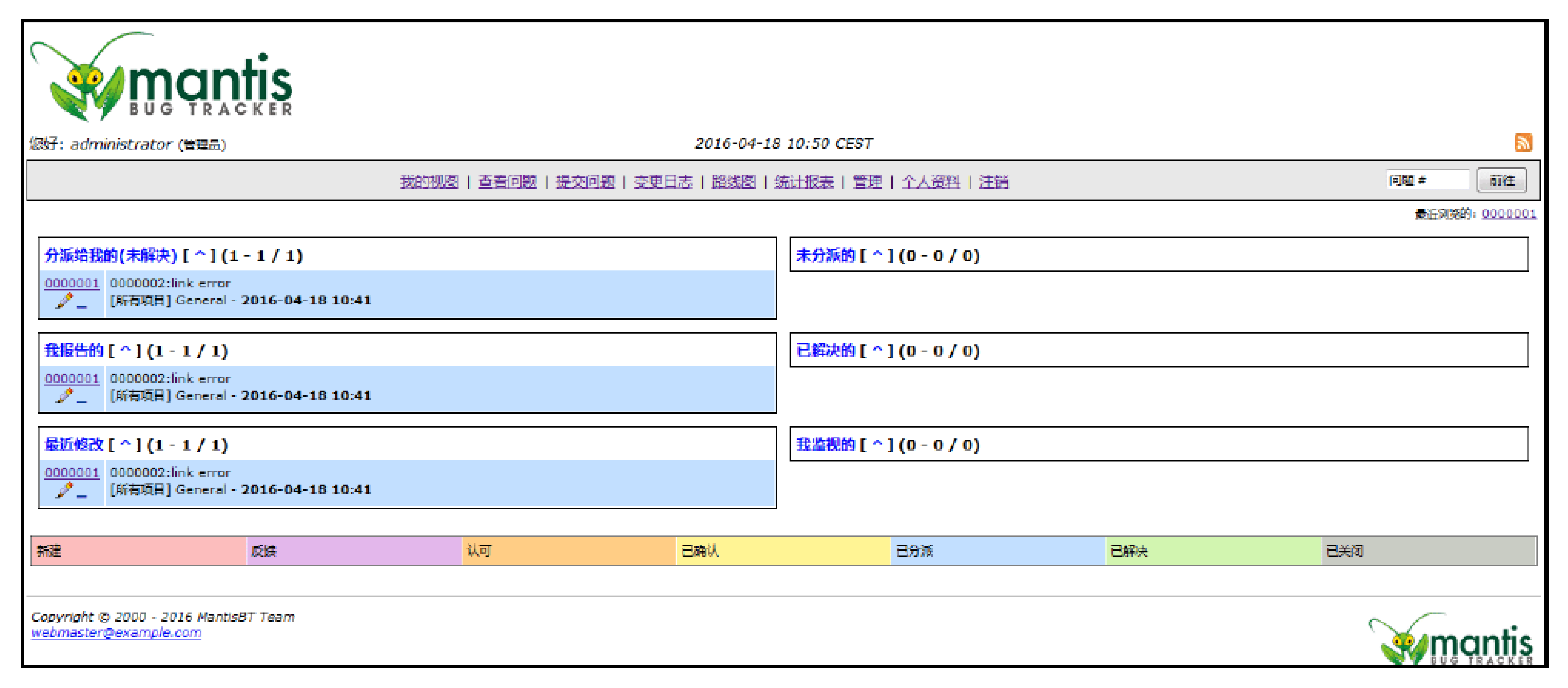


图 4-27　Mantis 的我的视图界面

查看问题

图 4-28 所示为查看问题(View Issues)界面。

图 4-28　Mantis 的查看问题界面

单击界面中的问题编号，可进入该问题的详细界面，并对该问题进行修改，如图 4-29 所示。

① “编辑”(Edit)：修改问题的各项基本属性，并添加注释。

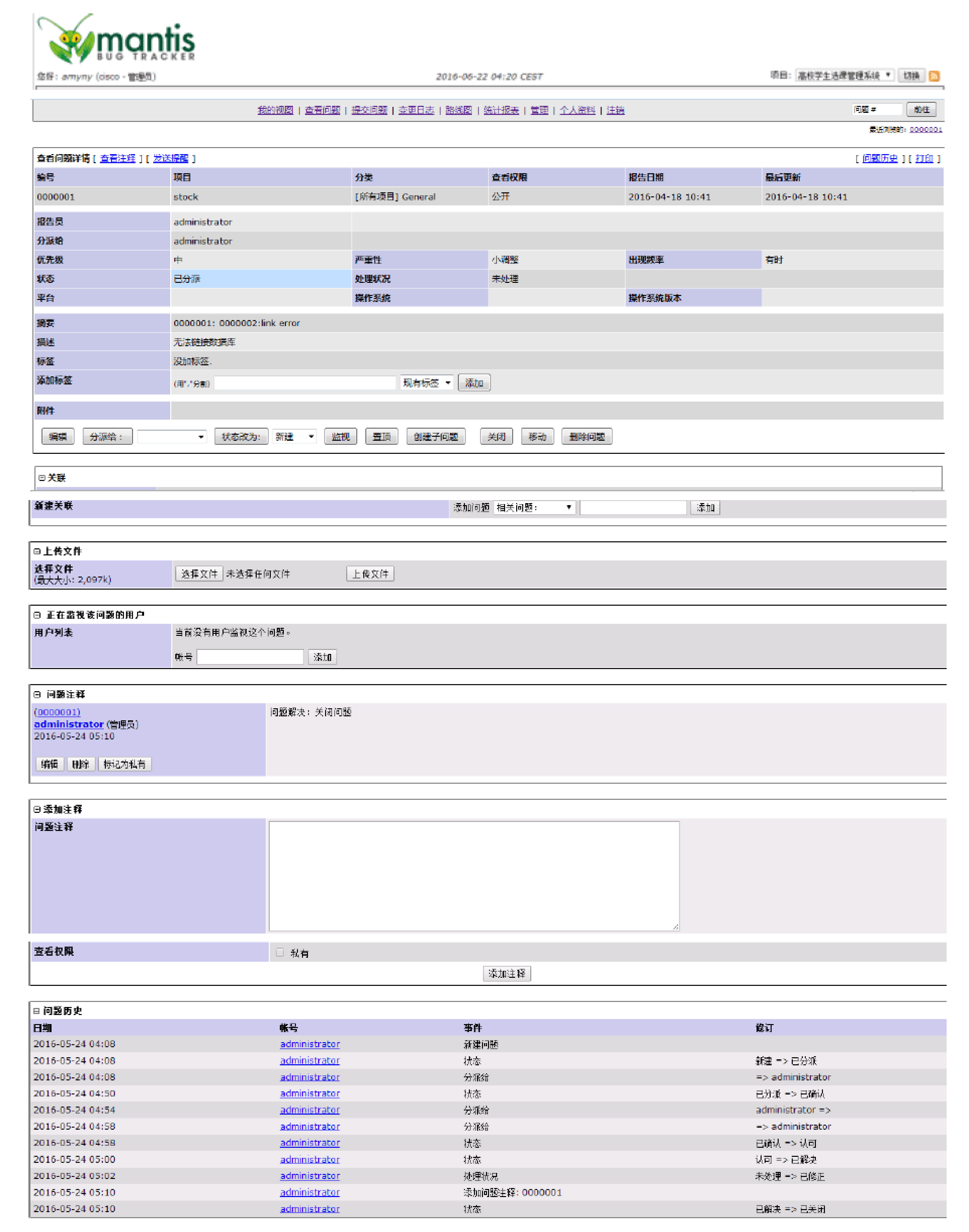


图 4-29　Mantis 的问题详细界面

② “分派给”(Assign To)：将问题分派给某个开发人员处理，分派之后系统将自动向被分派人发送邮件通知，被分派人打开 Mantis 之后将在“我的视图”界面上看到被分派的问题。

③ “状态改为”(Change Status to)：这里是指问题状态的转变，分为 6 个层次——新建、反馈、认可、已确认、已解决和已关闭。这是 Mantis 比较重要的一个功能，问题的每次变动都会发生状态的改变，以此来标记问题的处理情况。

④ “监视问题”(Monitor)：单击此按钮后，用户就可以对该问题进行监视。也就是说，只要该问题有改动，系统就会自动发邮件通知本人。这在“我的视图”界面上也可以体现出来。

⑤ “创建子问题”(Clone)：可以创建该问题的子问题。

⑥ “移动问题”(Move)：可以将该问题移动到别的项目中(需要相应的权限)。

⑦ “删除问题”(Delete)：删除无用的问题，已处理完毕的问题建议不必删除，关闭即可，以保留问题记录。

⑧ “关联”(Relationships)：可以指定问题之间的关联关系，具体关联方式见下拉菜单。

⑨ “上传文件”(Upload File)：可以上传与问题相关的文件，大小暂时限制为 5MB。

⑩ “问题历史”(Issue History)：此项为问题处理的历史记录。

提交问题

如图 4-30 所示，进入 Report Issue 界面，可以看到一个提交 bug 的表单，根据具体情况填写后提交即可。在提交报告时请注意，带*号的是必填项。界面上还提供了文件上传功能，只要是大小小于 2MB 的文件都允许上传，支持.doc、.xls、.zip 等格式的文件。这样在报告 bug 的时候，就可以上传相关的文件，为 bug 的解决提供更多的信息。全部填写完毕之后，就可以单击“提交报告”按钮来提交报告，之后系统会提示用户操作成功。返回“我的视图”界面，就可以看到新提交的报告。

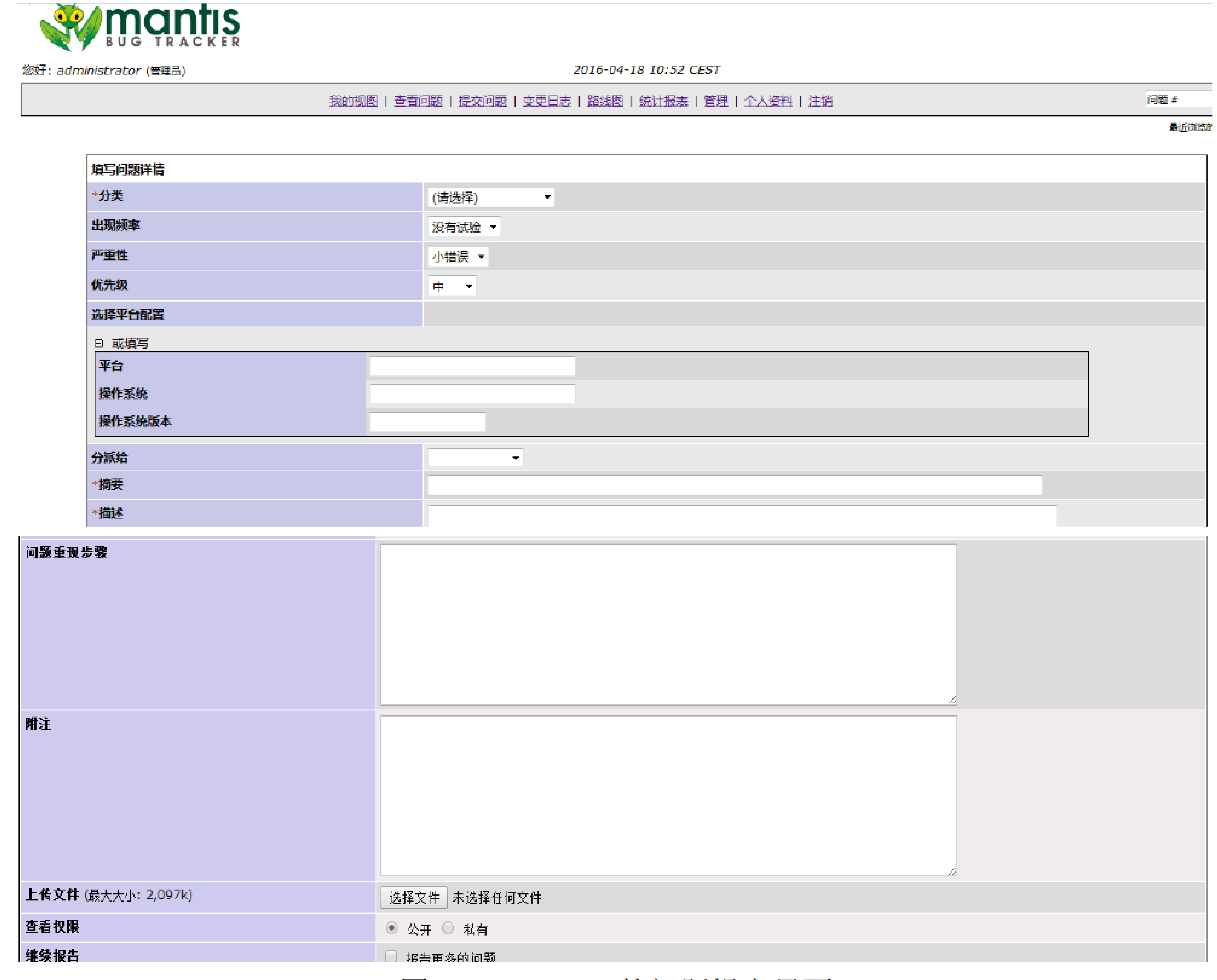


图 4-30　Mantis 的问题提交界面

修改日志

修改日志(Change Log)界面如图 4-31 所示。单击菜单栏中的“变更日志”选项，只有已经修改好了问题，需要给项目添加版本号，并且在提交/解决问题时都指定了相应版本号的日志才会显示。



图 4-31　Mantis 的日志修改界面

路线图

路线图(Roadmap)界面如图 4-32 所示。单击菜单栏中的“路线图”选项，如果开始没有指定项目，则首先进入项目选择界面，如果指定了默认值，则直接进入默认项目的路线图。其实，项目的路线图就相当于 bug 的日志信息，如图 4-32 所示，界面上列出了该项目下已解决的 bug 编号、所属组别、bug 摘要以及该项目产品的版本号变化，单击 bug 编号还可进入其详细信息界面。



图 4-32　Mantis 的路线图界面

个人资料

在个人资料(My Account)界面中可设置个人信息，包括密码、邮件、姓名；更改个人设置，可设置邮件通知的紧急程度级别等，根据个人需要和喜好来设置接收邮件通知的级别；管理平台配置，包括硬件平台；显示操作系统、版本等信息，如图 4-33 所示。

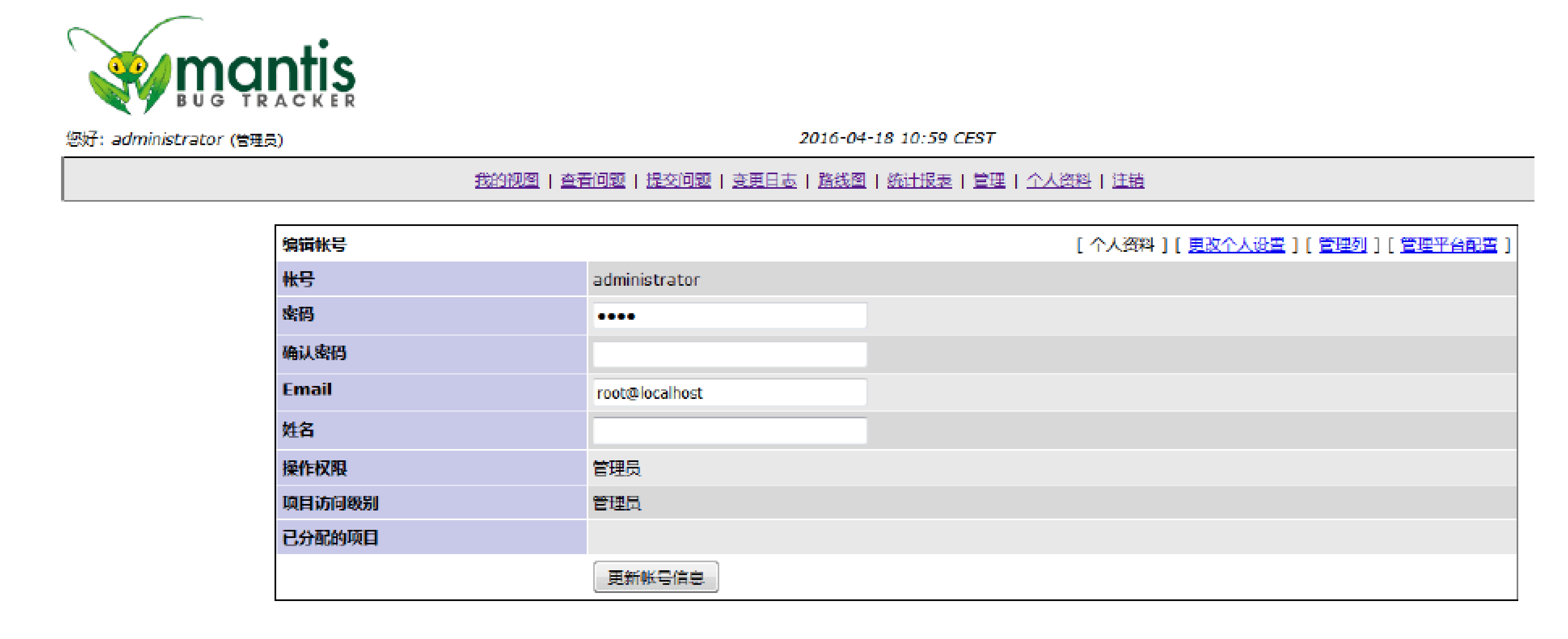


图 4-33　Mantis 的个人资料设置界面

统计报表

统计报表(Summary)界面如图 4-34 所示。单击“统计报表”选项，将会出现一个包含所有 Issue 报告的综合报表，界面上还提供了“打印报告”和“先进的摘要”等功能链接。

在这个综合报表中，按照 bug 报告详细资料中的项目，将所有的报告按照不同的分类进行了统计。这个统计报表有助于管理员及经理很好地掌握 bug 报告处理的进度，而且很容易就能把没有解决的问题与该问题的负责人、监视人联系起来，提高了工作效率。这个界面还提供了更多按不同要求分类的统计图表，如按状态统计、按优先级统计、按严重性统计、按项目分类统计和按处理状况统计。分别单击这 5 项，即可得到相应的统计图。单击“先进的摘要”链接，可进入刚才所显示数据的图形化报表界面，更便于查阅和对比。如果需要，还可以单击界面上方的“打印报告”链接，将所有的 bug 显示出来。如图 4-35 所示，在界面上列出了需要导出打印的 bug 列表，可以根据需要通过复选框选中需要打印的 bug，根据需要单击图标，bug 数据便相应地导出到该类型的文件里，实现打印输出需求。

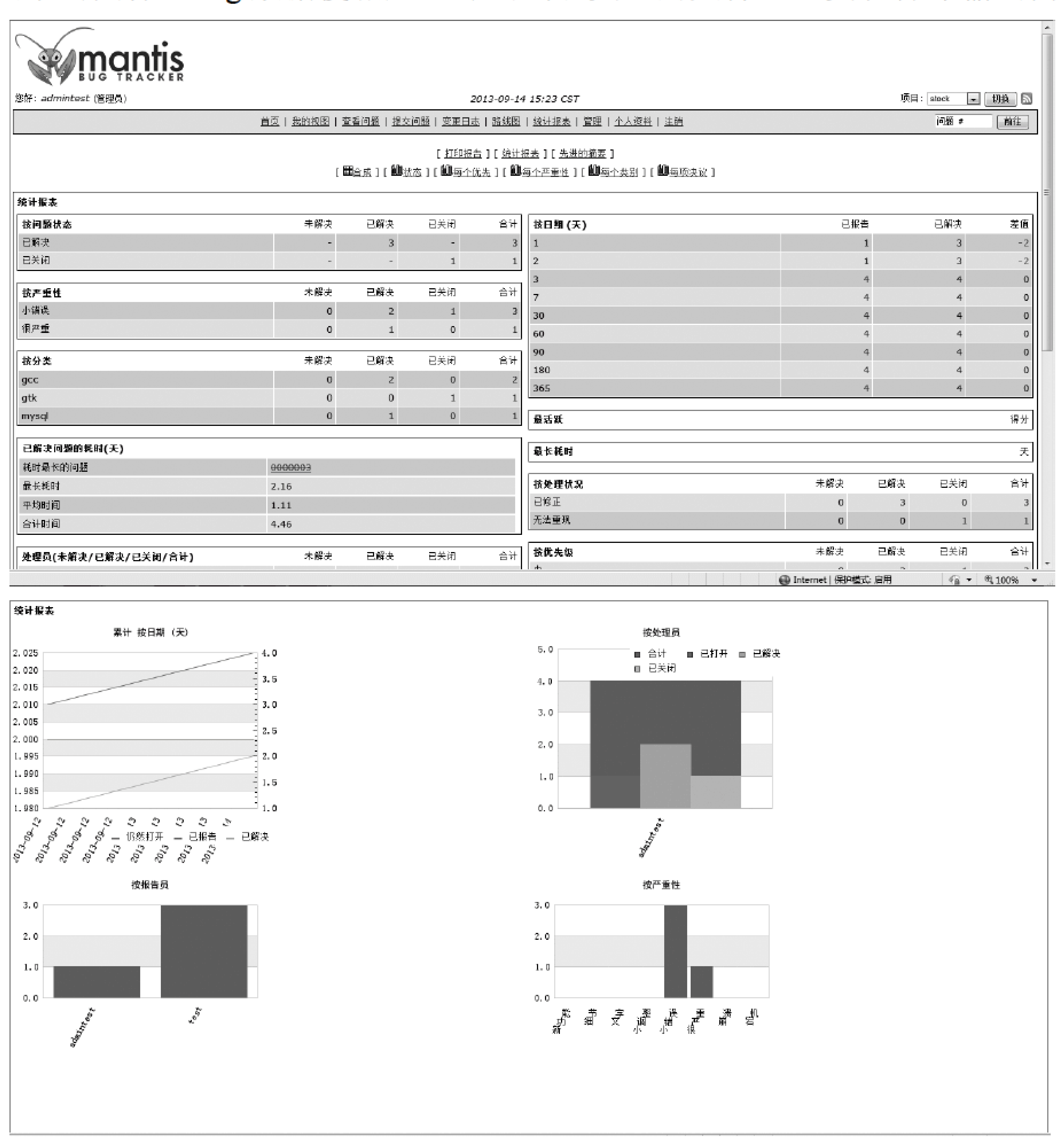


图 4-34　Mantis 的统计报表及统计图界面

MantisBT - stock

查看问题(1 - 2)

	P	编号	#	0	分类	严重性	状态	最后更新	摘要
☐	^	0000002			General	很严重	认可	2016-04-18	0000001:link error
☐	_	0000001			General	小调整	已分派 (administrator)	2016-04-18	0000002:link error

图 4-35　打印报告界面

管理

管理(Manage)界面如图 4-36 所示。单击菜单栏的“管理”选项，即可进入管理界面，管理界面包含用户管理、项目管理和自定义字段管理部分。

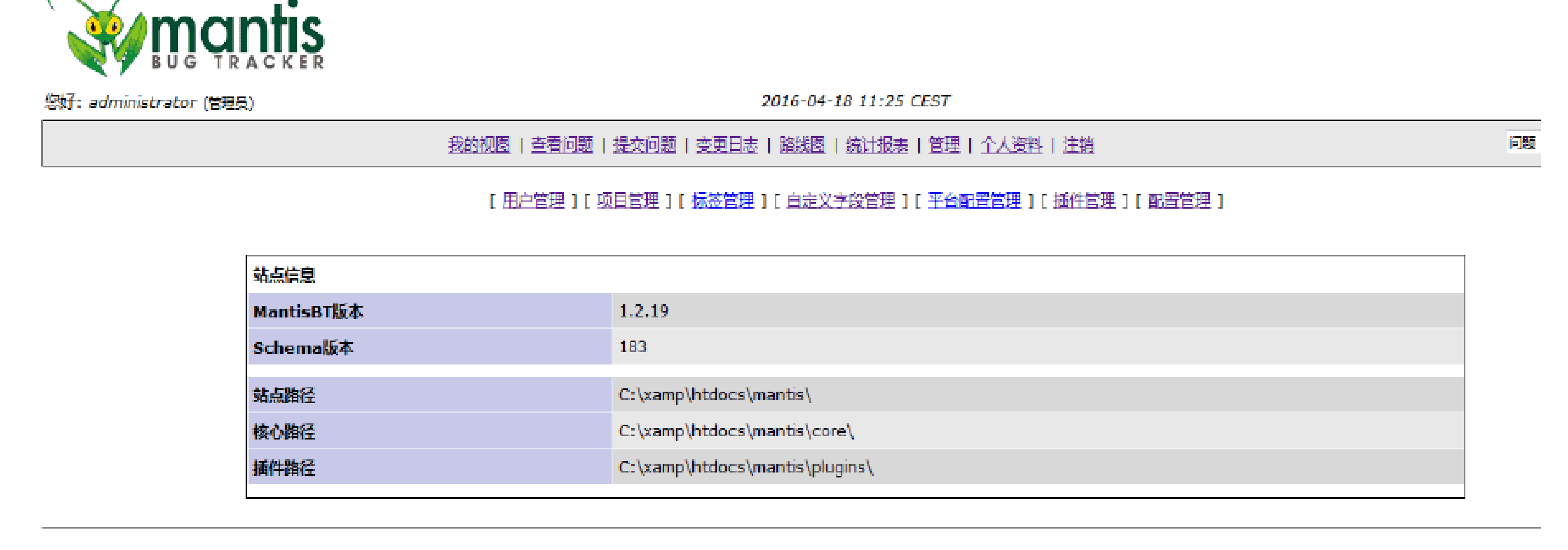


图 4-36　Mantis 的管理界面

① 用户管理为单击“管理”选项时进入的默认界面。

新账号：显示一周之内添加到该项目的新用户。

从未登录：显示目前存在却至今从未登录过的用户，对于此类用户可以通过单击“清理账号” 功能链接将其清除。

管理账号：在这里可以添加新用户和更新已有用户。单击 Create New Account，就可以添加新的用户，并指定其工作身份。单击现有账号名称，就可以对当前账号的资料进行更新，更新之后单击“重置”，系统就接受更新信息了。

② 单击“项目管理”，即可查看当前的所有项目。再次单击“创建新项目”选项，进入新项目创建界面，填好项目资料后单击“添加项目”，新的项目就被添加到系统中了。

注意：

上传文件时如果不指定存放路径，默认为系统路径。

③ 单击现有项目列表中的项目名称，就可以看到该项目的具体情况列表，列表中包括各个项目的名称、状态、查看状态以及说明列属性。在这个界面上可以进行以下操作：

编辑项目：在这里经理可以对项目的名称、状态、查看状态、上传文件的存放路径以及说明等内容进行更新。

子项目：可以创建属于该项目的子项目，或者指定某个项目为该项目的子项目。

增加分类：填入类别名称，单击“增加分类”便可在当前项目里增加类别。

编辑分类：单击“编辑”，进入类别编辑界面，还可以将当前的类别分配给指定的工作人员，这样会在该项目下提交一个新 bug 的时候，直接分派给指定的工作人员处理。

版本：可以对已有的项目版本进行更新或删除，也可以添加新的版本。

自定义字段：可以从已存在的自定义字段中选出所需要的，添加到项目的自定义字段里，也可以删除已添加的自定义字段。将自定义字段添加到项目后，在“提交问题”表单中会显示为必填字段。

添加用户至项目：将与项目相关的用户添加进来。

管理账号：对项目中所有相关人员的账号进行管理，可以删除那些在项目中不需要的账号。

④ 自定义字段管理是用于在提交问题的时候，若系统给予的填写项不满足实际需求，可以自行在这个功能界面上定义自己需要的字段，以便能更好地描述 bug 的情况，而且在过滤器里也会增加这个字段的属性，可以据其进行数据过滤。单击功能链接进入该界面即可新建自定义字段、修改已有的自定义字段。

2) 权限用户

经理

经理是整个软件开发过程中较为重要的管理人员。经理在系统下的使用权限比起管理员来说稍微低一些，由于前面已经详细说明了各个菜单功能的使用，因而这里主要说明经理在系统中所能使用的功能与管理员有什么不同。

对于前面提及的 10 个菜单功能，经理在使用的时候和管理员基本相同，但存在以下几个差异：

① 只能对自己的过滤器进行操作，对于管理员设为共有的过滤器只能使用而不能进行操作。

在“查看 Issue”的时候，可以通过复选框来对某条 bug 进行命令操作，经理级别的工作人员不能执行“删除”操作，系统会提示“你无权执行该项操作”。

② 在“管理”功能中，不能对用户进行管理，包括新建、删除用户等，也不能新建项目，只能管理现有项目的信息。

开发人员

开发人员就是负责整个软件开发的工作人员。使用 Mantis 这个 bug 跟踪管理系统，开发人员可以及时地发现和解决软件缺陷。在该系统中，开发人员的权限比起经理来说要稍低一些，从开始登录进入系统就能看出来，其主菜单栏的功能相对少了一些。比起经理和管理员的菜单栏，少了“统计报表”、“管理”这两个功能。

由于前面已经详细说明了各个菜单功能的使用，因而这里主要说明开发人员在系统中所能使用的功能与管理员和经理相比有什么不同。

开发人员只能设置私有的过滤器，可以共享管理员和经理创建的且属性设置为公有的过滤器。

在“查看 Issue”的时候，可以通过复选框来对某条 bug 进行命令操作，开发人员不能执行“删除”操作，系统会提示“你无权执行该项操作”。

修改人员

修改人员就是负责修改 Issue 的工作人员。修改人员的主菜单和开发人员的一样，但其使用权限还是比开发人员稍低一些，下面说明一下操作区别。

不能创建过滤器，但是可以使用由其他工作人员创建且属性被设为公有的过滤器。

在“查看 Issue”的时候，可以通过复选框来对某条 bug 进行命令操作，修改人员只能进行“复制”、“更新优先级”、“更新状态”、“更新视图状态”操作，对于其他的命令操作则无权执行。

在“我的视图”界面上，没有“分派给我的(未解决)”状态的 bug 数据列表，因为对于修改人员来说，不能将 bug 直接指派给他。因此，相应的界面上没有显示这类数据。

报告人员

报告人员就是专门负责提交 bug 报告的工作人员。报告人员的主菜单也和开发人员、修改人员的一样，但是其使用权限还是比修改人员稍低一些，下面具体说明一下操作区别：

① 不能创建过滤器，但是可以使用由其他工作人员创建且属性被设为公有的过滤器。

② 在“查看 Issue”的时候，对 bug 数据列表不能进行任何命令操作。

③ 在“我的视图”界面上，没有“分派给我的(未解决)”状态的 bug 数据列表，因为对于报告人员来说，也不能将 bug 直接指派给他。因此，相应的界面上没有显示这类数据。

查看人员

查看人员，顾名思义就是只具有查看权限的工作人员，在 Mantis 系统中，其权限最低，功能主菜单也相应更少。比起开发人员、修改人员、报告人员的主菜单功能来说，查看人员少了“报告 Issue”的功能，具体的操作区别是：因为查看人员为权限最低的工作人员，因此对于 Mantis 系统的操作功能基本上没有使用权限(除了个人资料的功能外)，而只能查看各个项目的 bug 流程及具体情况。

分派给我的工作

前面主要说明各个角色的工作人员在使用该系统的功能时所具备的权限，以下说明工作人员登录该系统后，该如何对分派的工作进行操作。

当工作人员登录系统并进入“我的视图”后，单击“分派给我的(未解决)”bug 数据列表的 bug 编号链接，进入分派任务的 Issue 界面。在该界面，Issue 根据各种情况被分成 7 个数据块来显示。在这里主要介绍查看问题详情数据块。

在分派任务的 Issue 界面，上面第一个数据块显示的就是 Issue 的详细资料，在这个数据表格中可以进行如下 13 种操作：

a) 查看注释：单击此链接，可直接跳转到该页面的注释数据。

b) 发送提醒：单击此链接，可对某个工作人员发送提醒，该提醒会直接在该工作人员的 Issue 监视视图里生成，并在该 Issue 的注释里也增加一条记录。注意：这个功能除了查看人员之外，其他工作人员也可以使用。

c) Issue 历史：单击此链接，将直接跳转到该界面的 Issue 历史数据。

d) 打印：单击此链接，将直接在界面上生成这个 Issue 的详细数据，可以通过 IE 提供的打印功能进行打印。

e) 修改 Issue：如果 Issue 的信息需要修改，则可以单击“编辑”按钮，直接进入 Issue 修改界面。根据实际情况进行修改，然后单击“更新信息”按钮。如果不需要修改了，可以单击“返回 Issue”回到前面的界面。需要注意的是：修改 Issue 的功能只有管理员、经理、开发人员和修改人员能使用。

f) 分派给：这个功能可以把当前的 Issue 直接指派给选定的工作人员。注意：指派功能只有管理员、经理、开发人员和修改人员能使用，而且只有管理员才能指派给自身。

g) 将状态改为：这个功能可以对当前的 Issue 流程状态直接进行修改。注意：该功能只有管理员、经理和开发人员可以使用。

h) 监视 Issue：这个功能可以把当前这个 Issue 置于所监视的 Issue 范围之内。注意：这个功能除了查看人员之外，其他工作人员也可以使用。

i) 置顶：这个功能可以将当前这个 Issue 显示在所有“分派给我的(未解决)”Issue 中

的第一行。

j) 创建 Issue 子项：这个功能主要用于创建与当前 Issue 相关的 Issue，单击该按钮进入如图 4-37 所示的界面。界面上用红框标注的就是设定创建 Issue 子项后和当前 Issue 的关系。同时，在“关联”数据块中会增加一条记录。注意：创建子项功能只有管理员、经理、开发人员和修改人员才能使用。

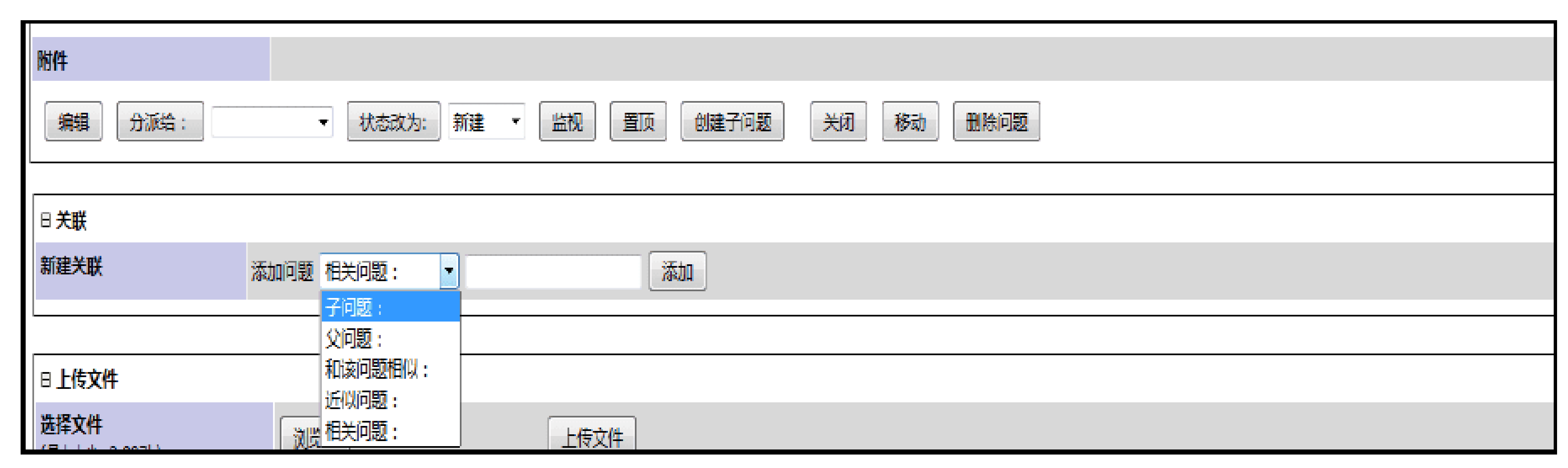


图 4-37　Mantis 的子问题创建操作

k) 关闭：这个功能将该问题关闭，不再修改、讨论。注意：关闭功能只有管理员、经理才能使用。

m) 移动 Issue：单击此链接可直接将该 Issue 转移到别的项目中。注意：移动 Issue 功能只有管理员、经理和开发人员才能使用。

n) 删除 Issue：单击此链接将直接删除该 Issue，只有管理员才有这个权限。

4.4.3　Mantis 应用举例

以对高校学生选课系统进行软件测试为例，简单介绍一下 Mantis 的使用过程。

1. Mantis 的应用过程举例

打开 IE，输入 http://localhost/mantis 并按 Enter 键，应该就可以看到 Mantis 的登录页面了。可以用默认用户名 administrator 和密码 root 登录进去，进行管理设置。

Mantis 目录下有一个 admin 目录，如果在 IE 中打开这个目录下的 index.php 文件进行查看，就会知道这个目录用于进行 Mantis 管理，使用这个模块可以检查 Mantis 是否安装完全，对旧版本的 Mantis 进行升级，以及对 Mantis 的页面 CSS 文件进行修改。使用这个管理模块时是不需要用户名和密码的，因此任何人都可以通过这个管理模块来查看 Mantis 的系统信息。而且由于有升级模块，在这里还可以直接对数据库进行修改。因此，如果被未授权的人打开，就可能出现不好的结果。最好按照系统给出的建议，在配置完成后将这个 admin 目录删除。注意一定是删除而不是改名。改名后仍然是可以访问的。在添加管理员用户后，删除系统默认的 administrator 用户。

1) 创建项目

进入 Mantis 项目管理界面(如图 4-38 所示)，从菜单中选择“管理”，再选择“项目管理”。

(1) 添加项目：单击“创建新项目”按钮，本例以高校学生选课系统为被测软件，读者也可以自选项目。在“项目名称”文本框中输入项目名称“高校学生选课系统”，其他保持默认即可，如图 4-39 所示。

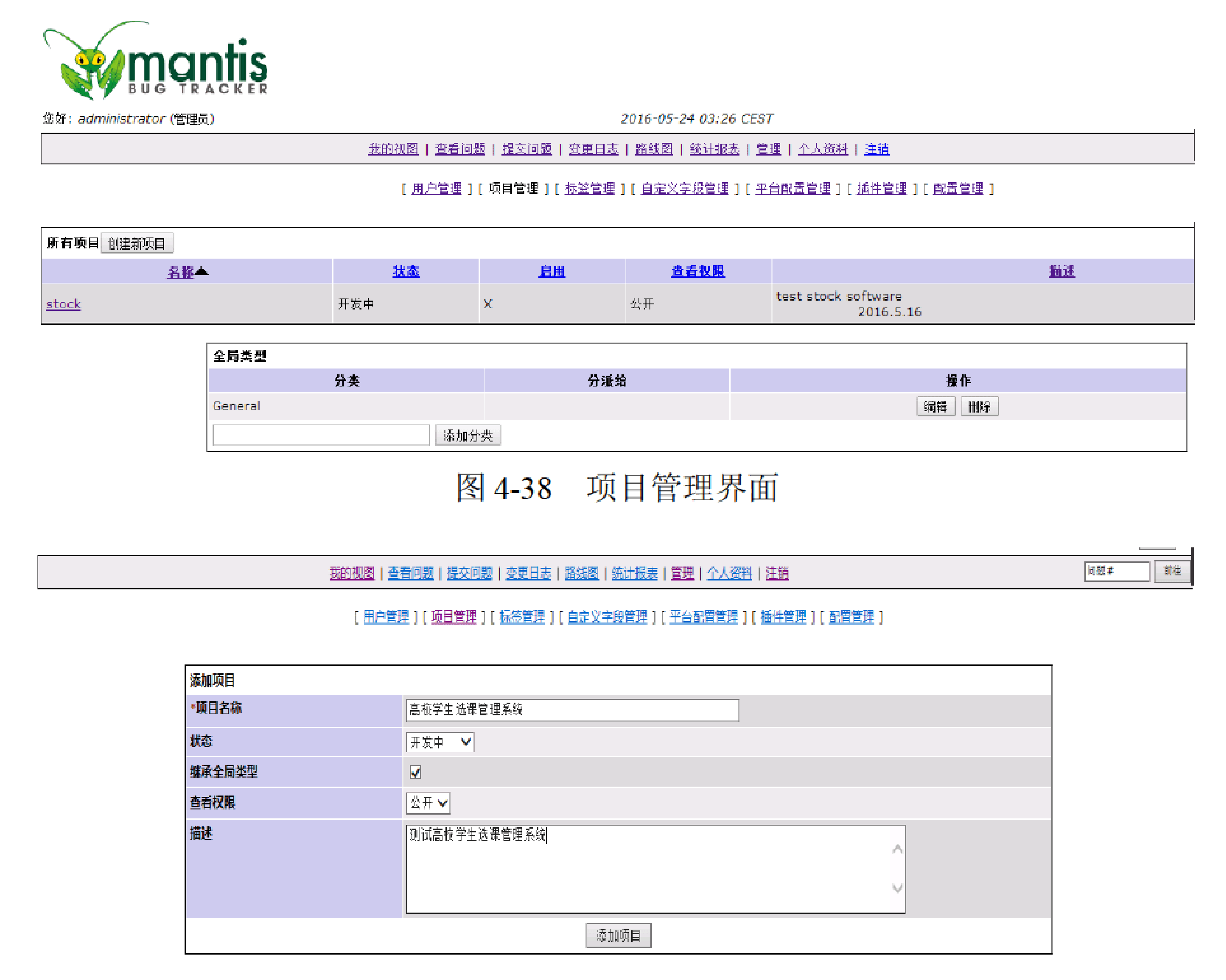


图 4-38　项目管理界面

图 4-39　项目创建结果

(2) 添加分类：单击新添加的项目“高校学生选课系统”，在分类中“添加分类”，如图 4-40 所示。

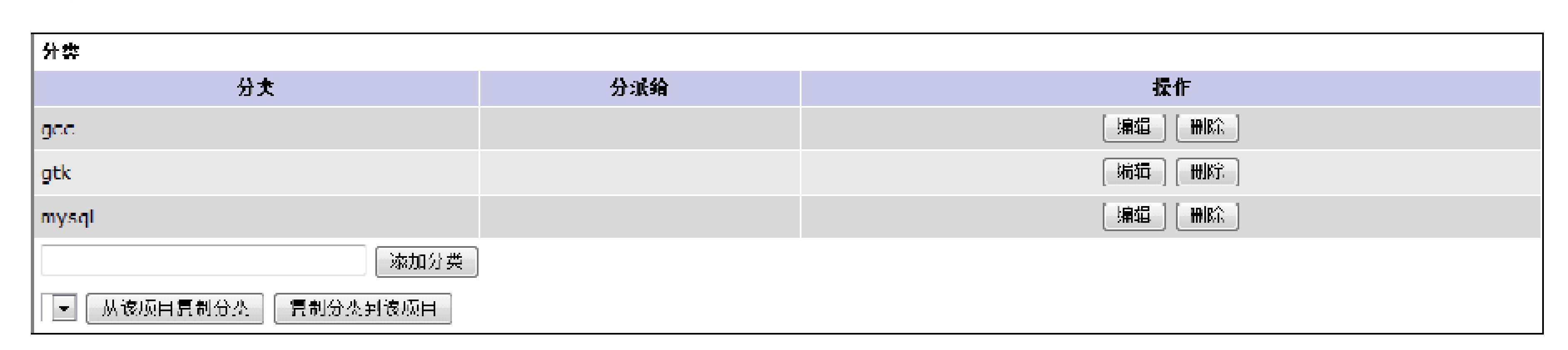


图 4-40　添加分类

2) 提交问题

(1) 选择项目：单击“提交问题”进入如图 4-41 所示界面，选择提交问题所属的项目(角色为报告员)。

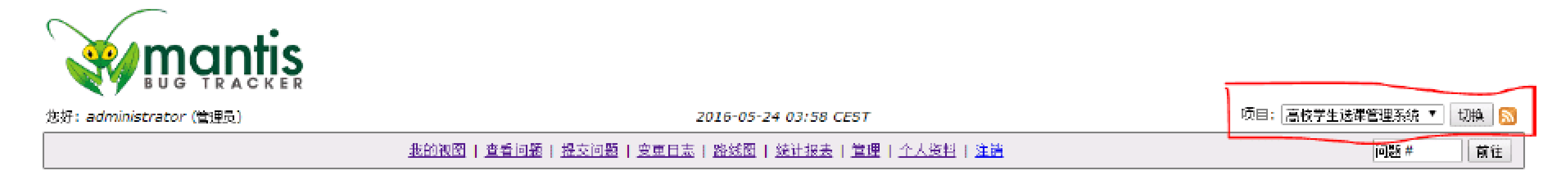


图 4-41　选择项目

(2) 填写项目详细资料：单击“选择项目”按钮，进入提交问题主界面，填写项目的详细资料，完成后单击“提交报告”，如图 4-42 所示。

图 4-42 中有些选项是打了星的，表示这些是必填内容。填好问题报告后，单击“提交报告”，就可以将此问题提交到系统，系统将通过 e-mail 通知项目组的相关人员。

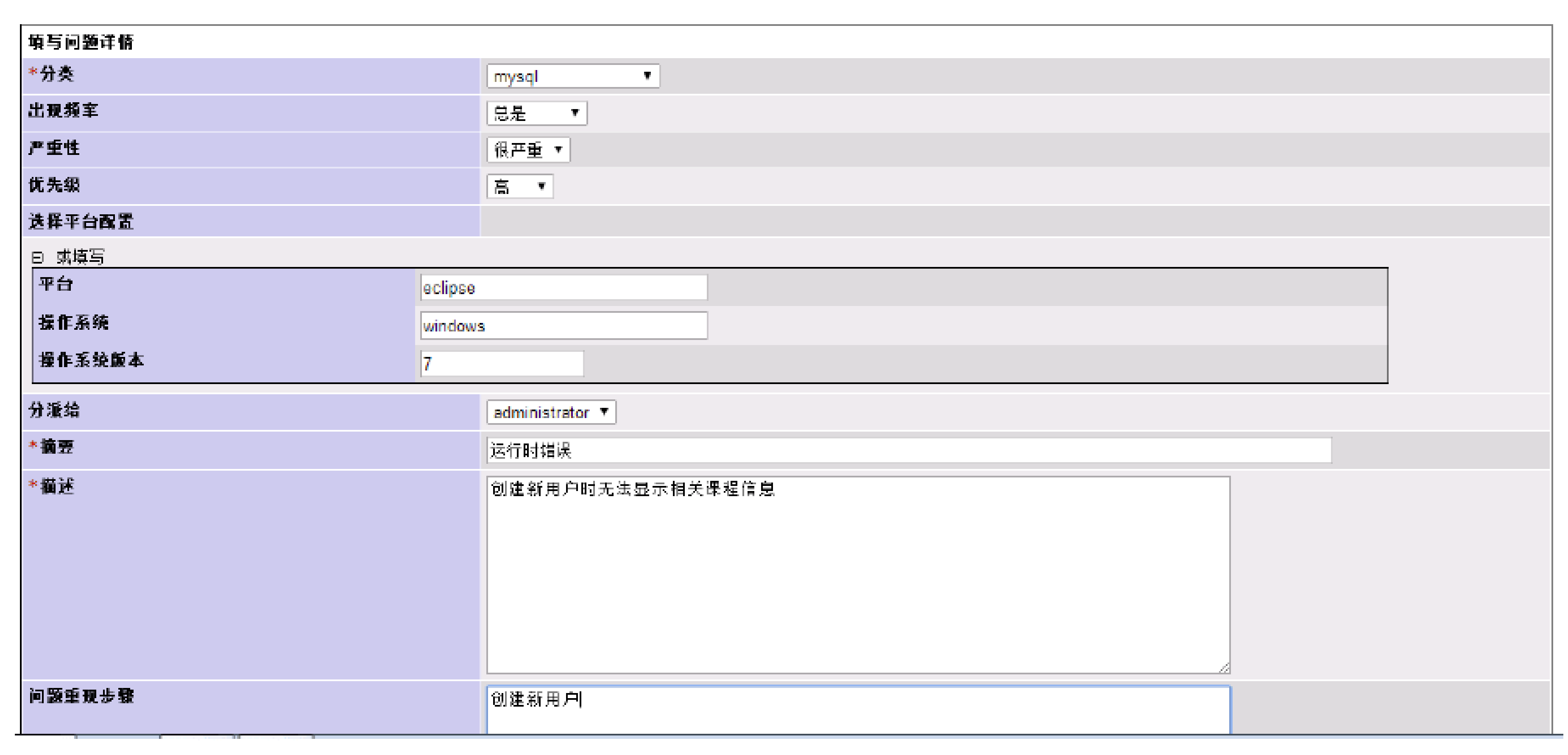


图 4-42　填写项目详细资料

3) 查看问题

只需要单击工具栏上的“查看问题”，就可以看到刚刚创建的问题，如图 4-43 所示。

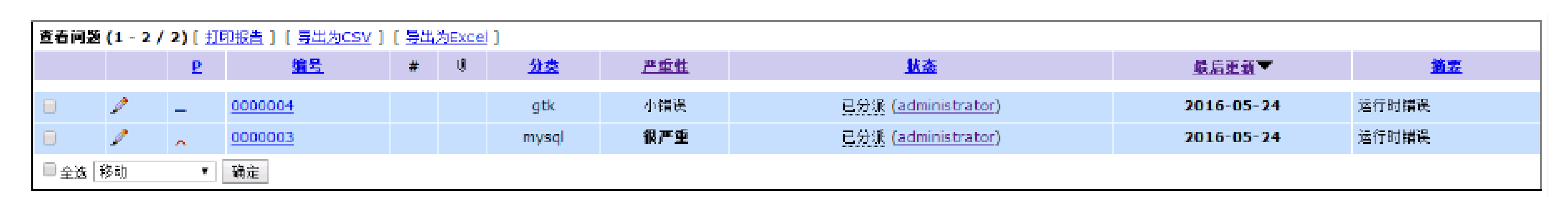


图 4-43　问题查看列表

4) 更新问题

单击问题编号进入问题详情界面，单击“编辑”，对问题进行更新，更新信息后，单击“更新信息”，如图 4-44 所示。问题更新后，会给提交问题的报告员发送一封包含更新信息的邮件。

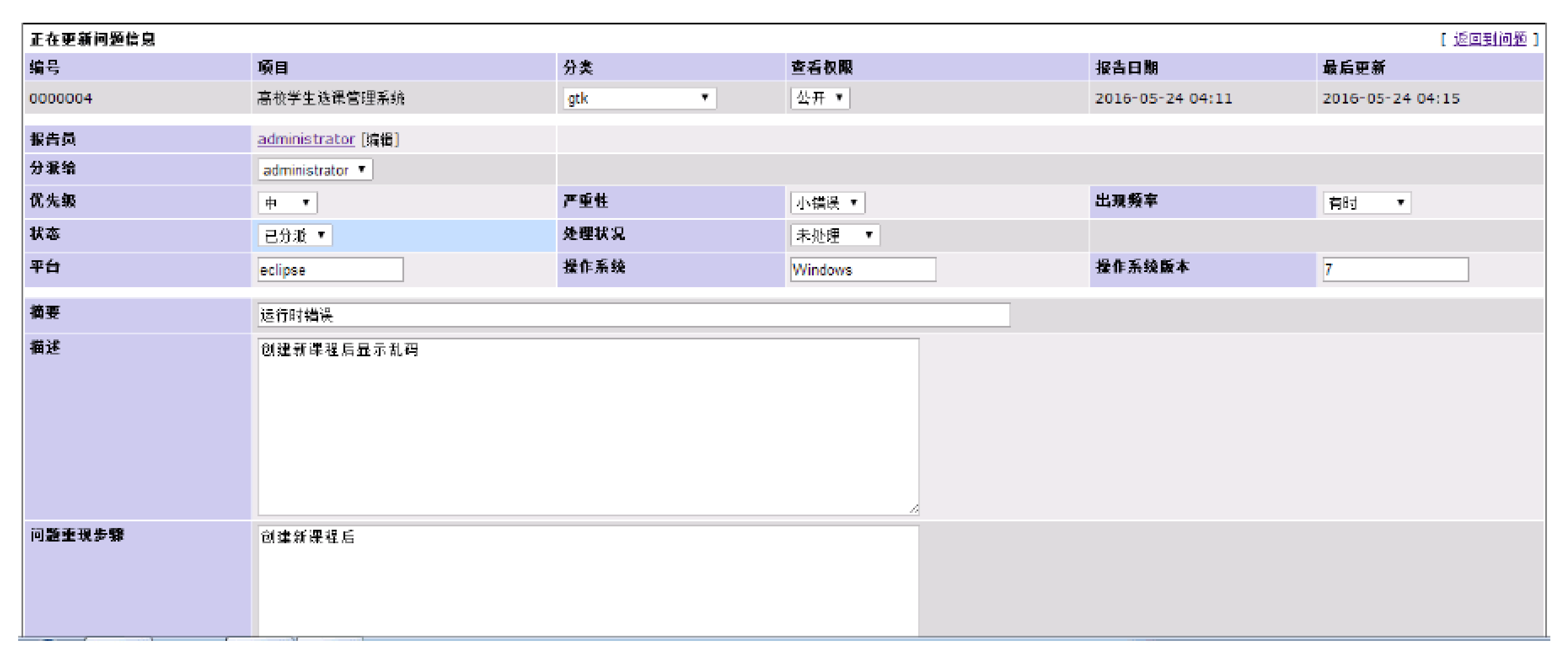


图 4-44　填写问题更新信息

(1) 查看修改结果，如图 4-45 所示。

(2) 查看问题更新历史，如图 4-46 所示。

5) 创建自定义字段

对创建的字段进行修改，如图 4-47 和图 4-48 所示。

6) 查看最后的缺陷情况

缺陷情况列表如图 4-49 所示。

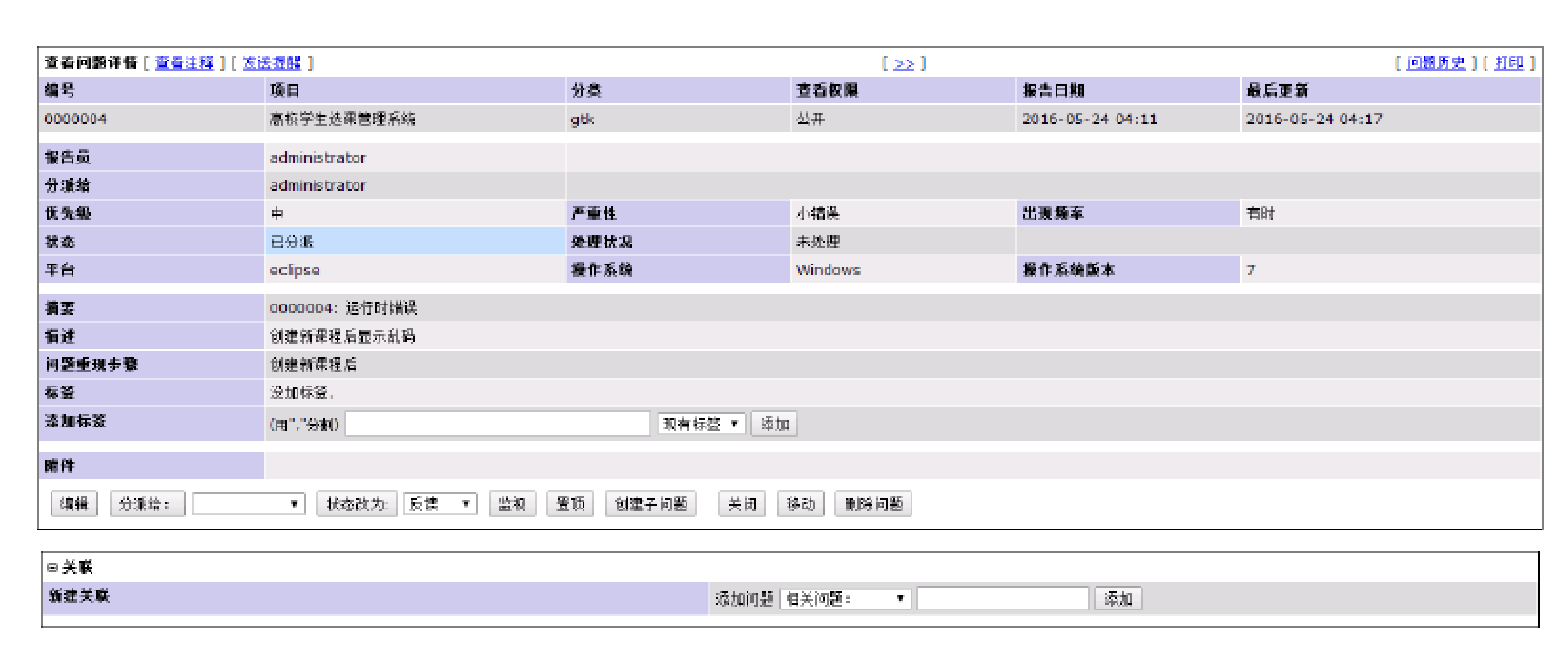


图 4-45　查看修改结果

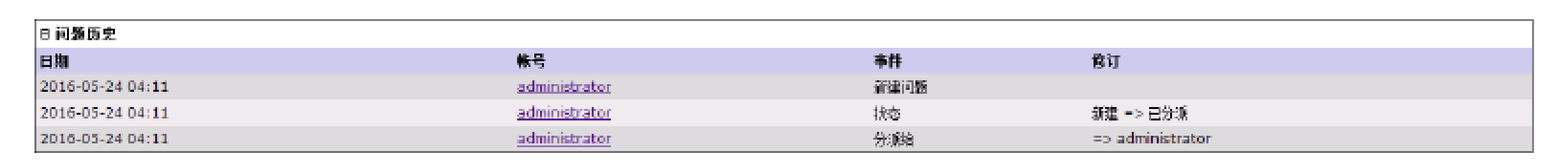


图 4-46　查看问题更新历史

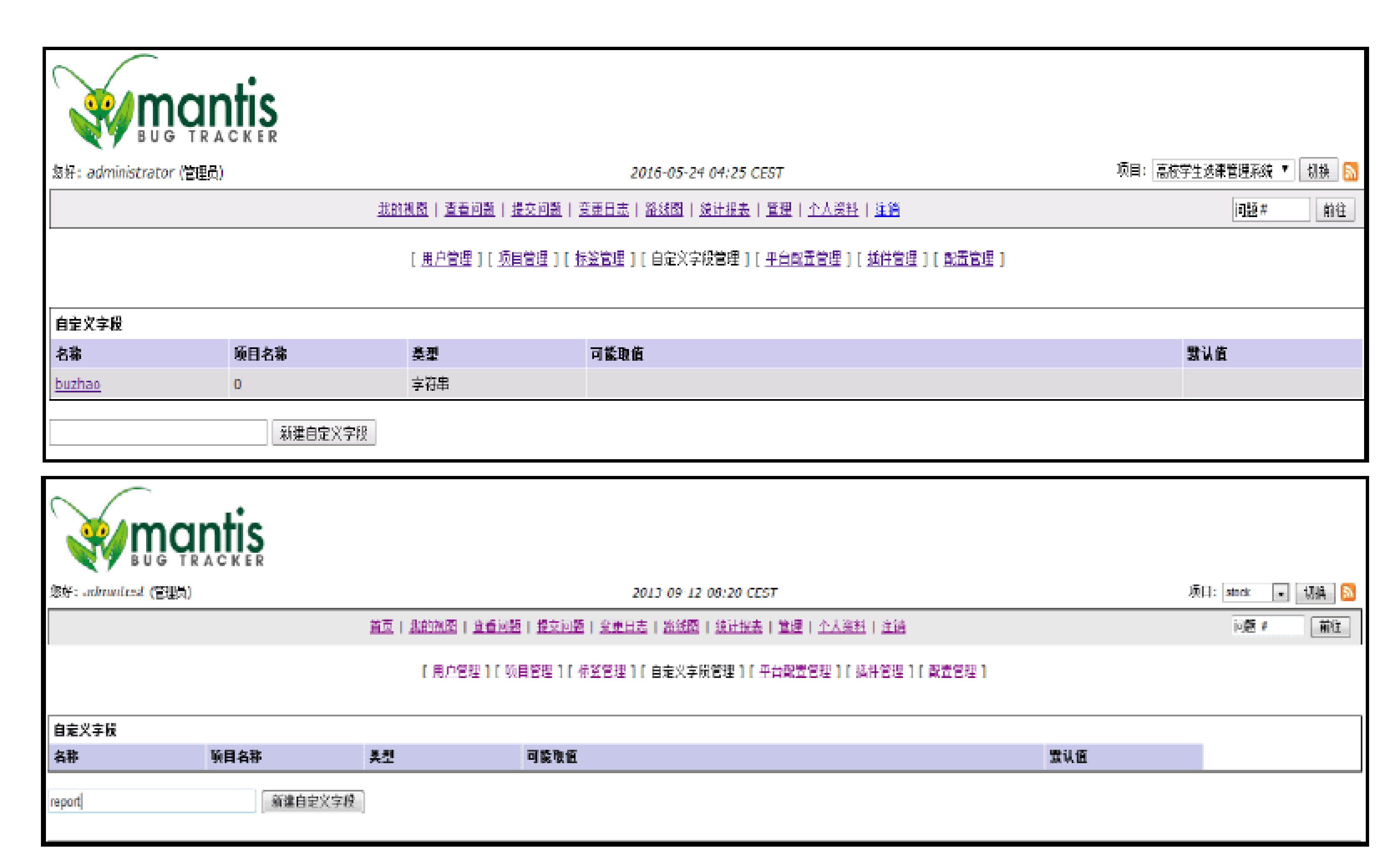


图 4-47　创建、修改字段并查看结果



图 4-48　创建、修改字段并查看结果

[用户管理] [项目管理] [标签管理] [自定义字段管理] [平台配置管理] [插件管理] [配置管理]

修改自定义字段	
名称	report
类型	字符串
可能取值	
默认值	
正则表达式	
读权限	复查员
写权限	复查员
最小长度	0
最大长度	0
添加到过滤器	☑
在创建问题时显示	☐
在更新问题时显示	☑
解决问题时显示	☐
关闭问题时显示	☐
报告问题时必需	☐
修改问题时必需	☐
解决问题时必需	☐
关闭问题时必需	☐

修改自定义字段

删除自定义字段

关联自定义字段到项目	
关联项目:	
所有项目:	stock 高校学生选课管理系统
顺序:	0

关联自定义字段

自定义字段

名称	项目名称	类型	可能取值	默认值
report	1	字符串		

图 4-48(续)

分派给我的(未解决) [^] (1 - 2 / 2)

0000004	运行时错误 gtk - 2016-05-24 04:17
0000003	运行时错误 mysql - 2016-05-24 04:08

未分派的 [^] (0 - 0 / 0)

我报告的 [^] (1 - 2 / 2)

0000004	运行时错误 gtk - 2016-05-24 04:17
0000003	运行时错误 mysql - 2016-05-24 04:08

已解决的 [^] (0 - 0 / 0)

最近修改 [^] (1 - 2 / 2)

0000004	运行时错误 gtk - 2016-05-24 04:17
0000003	运行时错误 mysql - 2016-05-24 04:08

我监视的 [^] (0 - 0 / 0)

新建	反馈	认可	已确认	已分派	已解决	已关闭

图 4-49　缺陷情况列表

7) 统计报表

单击工具栏上的“报表统计”，以表格的形式对问题进行统计，可按问题状态、按严重性、按分类等进行统计，如图 4-50 所示。

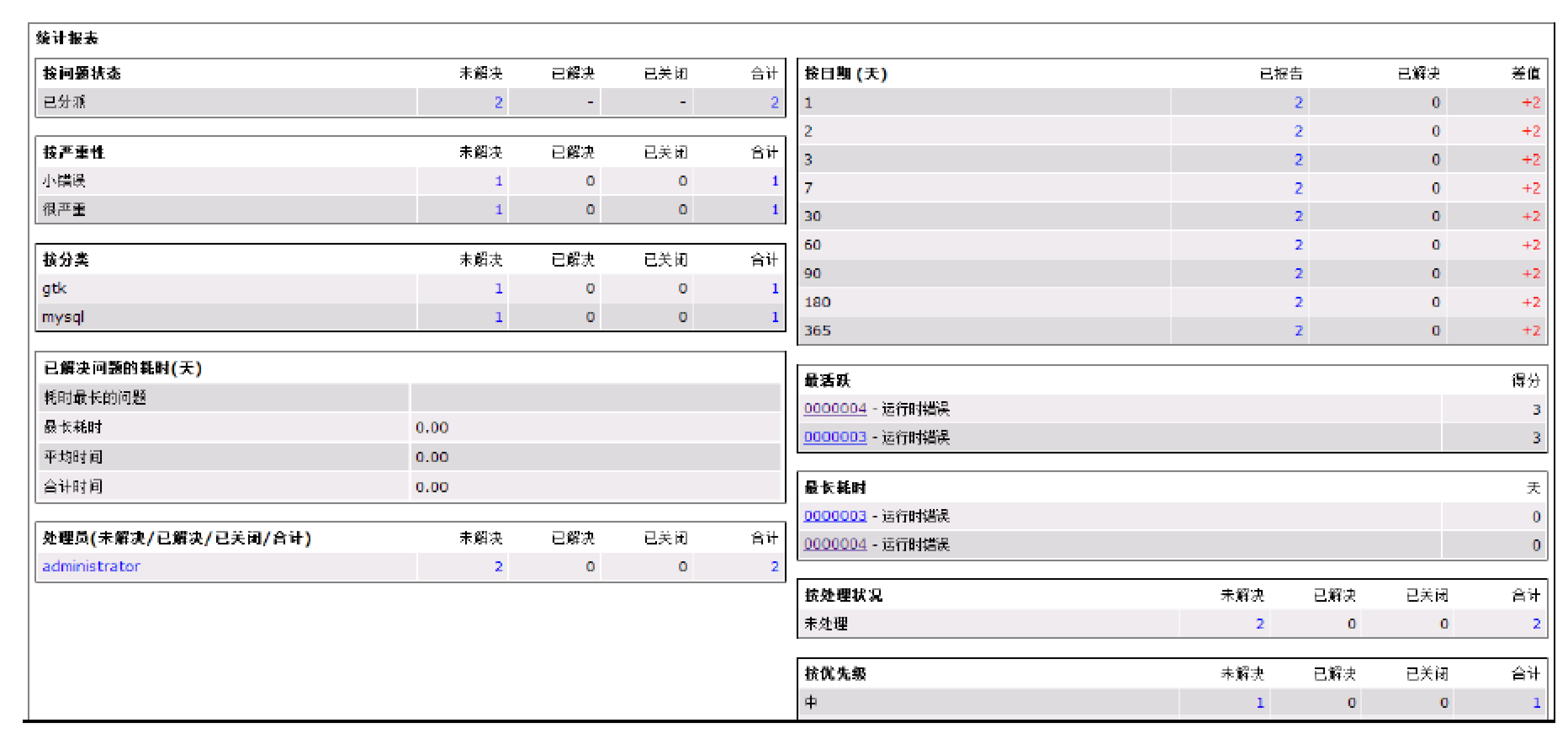


图 4-50　缺陷统计

2. 高校学生选课管理系统中的缺陷处理流程举例

(1) 管理员创建项目之后，项目经理 admin 对测试项目(高校学生选课管理系统)进行编辑，如图 4-51 所示。



图 4-51　编辑、测试项目

(2) 添加分类，还可以设置、修改版本信息，如图 4-52 所示。

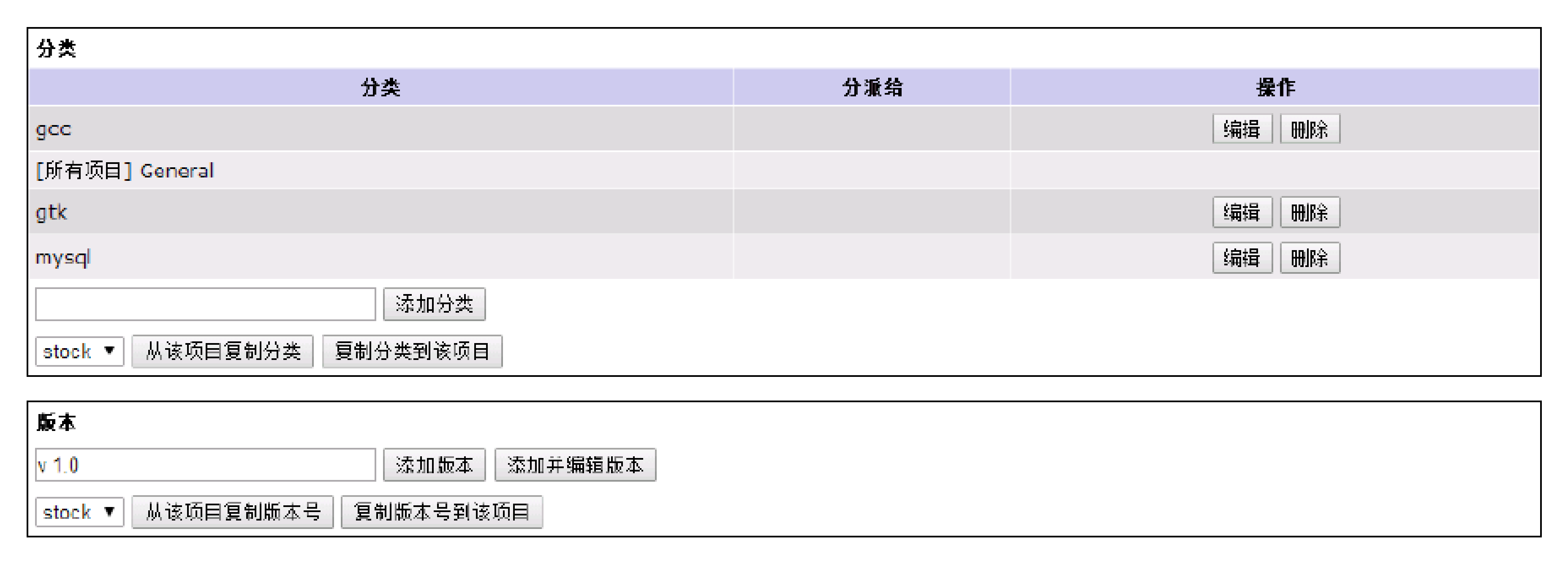


图 4-52　添加测试分类及版本信息

(3) 测试人员 kerry(报告人员)发现问题(高校学生选课管理系统在安装编译的时候发生问题，软件终止且不能继续运行)，编写缺陷报告后提交：高校学生选课管理系统出现缺陷。缺陷状态自动设置为“新建，如图 4-53 所示。

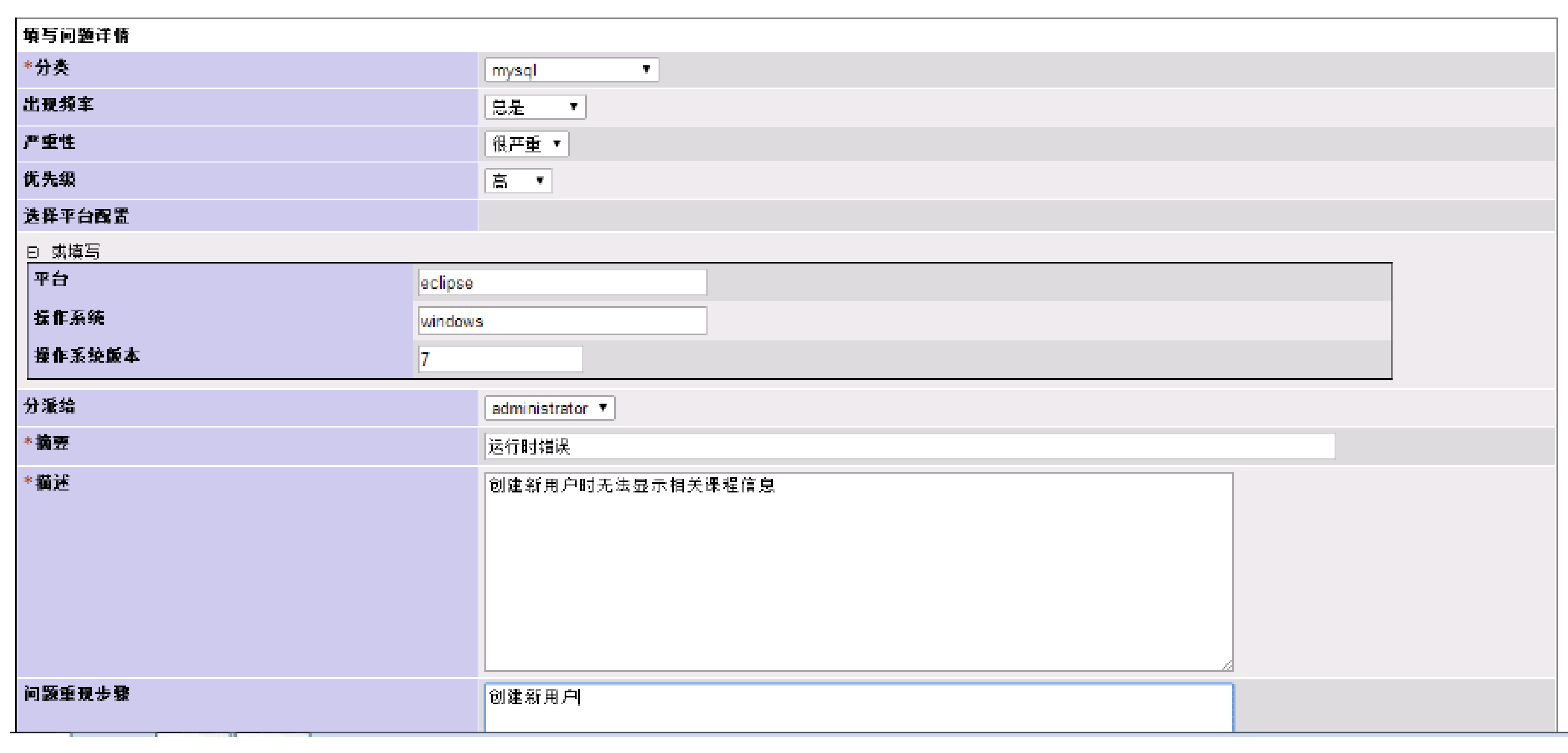


图 4-53　编写问题报告

(4) 开发人员 amyny 登录后在查看问题界面上看到状态为“新建”的 bug 后，打开问题报告详细界面，按照问题重现步骤实现 bug，发现 bug 可以重现，将缺陷状态改为“已确认，如图 4-54 所示。



图 4-54　将缺陷状态设置为“已确认”

(5) 项目经理审查后，表示对该 bug 认可，将缺陷状态设置为“认可”，并将其分派给开发人员 amyny，如图 4-55 所示。



图 4-55　将缺陷状态设置为“认可”并分派

(6) amyny 发现分派给自己的问题，将问题解决后更新缺陷报告(说明缺陷已经被解决，并说明软件的现状)，并更新缺陷状态为“已解决”，如图 4-56 所示。



图 4-56　将缺陷状态设置为“已解决”

(7) kerry 发现 bug 已经被修复，对该 bug 进行验证。若验证未通过，可以重启问题；若通过验证，不进行任何操作，如图 4-57 所示。



图 4-57　对缺陷进行验证

(8) 项目经理发现问题被解决，且未被重启，将该问题关闭，如图 4-58 所示。



图 4-58　关闭问题

(9) 现在任何级别的用户查看问题时，都将发现该问题已经不存在了，如图 4-59 所示。

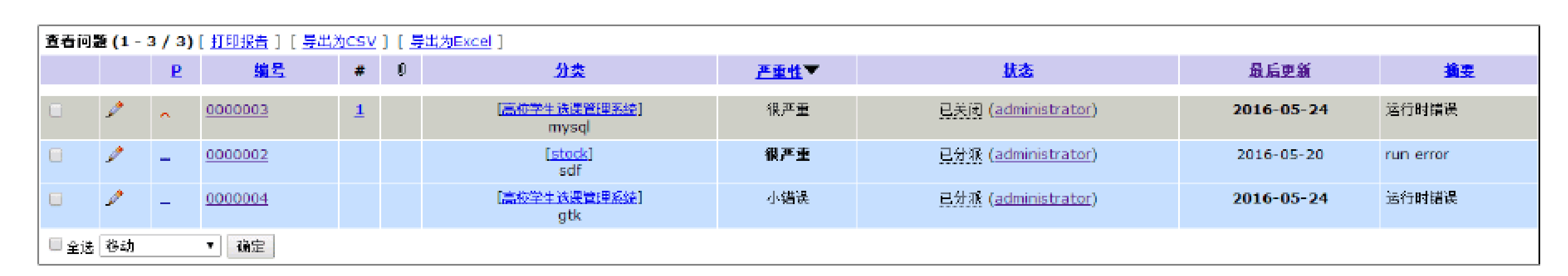


图 4-59　问题查看界面上已经没有该问题了

习题和思考题

1. 什么是软件缺陷？我们一般是如何描述和分类软件缺陷的?

2. 简述缺陷的来源与影响，分析缺陷都是在软件开发生命周期的哪个阶段产生的？

3. 什么是软件缺陷管理，缺陷管理报告单包括哪些内容？具有什么特点？

4. 根据自己的理解，画出软件缺陷管理流程图，并解释软件缺陷管理流程图的关键要素。

5. 我们如何度量软件缺陷？如何分析软件缺陷？如何对软件缺陷进行统计？

6. 简述软件缺陷描述中的缺陷基本信息和软件缺陷分类中的缺陷属性。

7. 软件缺陷报告所包含的主要内容有哪些？其撰写标准主要有哪些？

8. 建立 Mantis 缺陷管理环境，并通过一个具体的实例，详细说明 Mantis 缺陷管理的功能和流程。

9. 选择 Bugzilla、BugOnline 或其他缺陷管理工具，并将它们与 Mantis 进行比较，评判它们对缺陷管理流程的支持及各自的特点。

第5章　基于生命周期的软件测试方法

如同软件生命周期，我们也可以将软件测试阶段按照软件生命周期去划分，形成基于生命周期的软件测试(后面简称生命周期测试)。此时，软件测试可以划分为：测试需求分析、测试计划、测试设计、测试开发、测试执行、测试评估。这样，生命周期测试方法将测试延伸到了需求分析、设计审查活动中，也就是将质量保证的部分活动归为测试活动，真正体现了尽早地和不断地进行软件测试的原则，确保了对软件生命周期的每个阶段进行质量管理，并通过测试手段实现对各个阶段的质量保证。

5.1　生命周期测试概念

按照传统的软件生命周期的观点，测试是在编程活动之后进行的，是软件开发的最后一个阶段。随着人们对软件工程化的重视以及软件规模的日益扩大，软件分析、设计的作用越来越突出，而且有资料表明，60%以上的软件错误并不是程序错误，而是需求分析和系统设计错误。比如，从 IBM 提供的数据来看，对于一个大约 60 个缺陷/千行的软件，2/3 的缺陷产生在需求和设计阶段。而对于在需求和设计阶段发现的缺陷，修正的花费最小；否则，到了系统测试阶段再修正所发现的缺陷，花费是以上的 10 倍，到了产品发布阶段才修正所发现的缺陷，花费将是 100 倍。这说明，若能在需求和设计阶段就发现软件的缺陷，修正所需的花费将比在编程完成后再进行测试所需的花费少很多。因此，做好软件需求和设计阶段的测试工作就显得非常重要，这就使得传统的测试概念扩大化，形成了生命周期测试概念。

生命周期测试应伴随整个软件开发周期，此时测试的对象不仅仅是程序，需求、功能和设计同样要测试。比如：在项目需求分析阶段就要开始参与，审查需求分析文档、产品规格说明书；在设计阶段，要审查系统设计文档、程序设计流程图、数据流图等；在代码编写阶段，需要审查代码，看是否遵守代码的变量定义规则、是否有足够的注释行等。测试与开发同步进行，有利于尽早地发现问题，同时缩短项目的开发建设周期。

5.1.1　生命周期测试的工作划分

生命周期测试意味着测试与软件开发平行，在软件开发的所有阶段进行测试，确保在尽可能早的阶段点去修正缺陷，用来减少测试成本。与软件开发一样，生命周期测试需要正式的测试流程来支持。即在软件开发团队组建时，测试小组也同时建立，在项目开始时，

测试计划和测试条件也随着开始，并在生命周期的各阶段结束点测试系统，以确保能正确地开发系统，以及尽可能在生命周期最早的可能点发现软件缺陷，如图 5-1 所示。

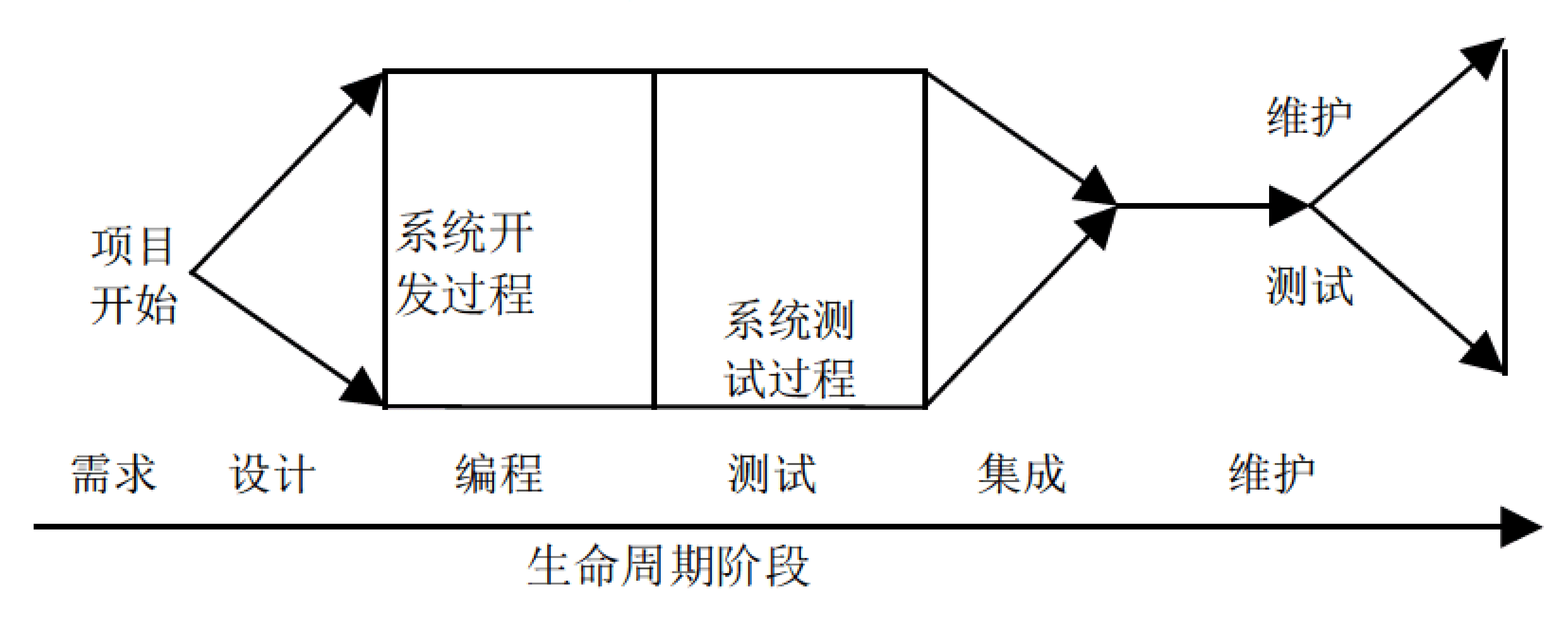


图 5-1　生命周期测试概念

图 5-1 表示当项目开始时，系统开发过程和系统测试过程同时开始，开发小组开始系统开发过程，而系统测试小组开始计划系统测试过程。两个小组在同一点开始，使用相同的信息。测试小组在开发过程的若干个预定点对系统进行连续的测试，检查开发过程的结果。软件生命周期中要进行的测试如表 5-1 所示。

在需求阶段，重点是确认定义的需求符合机构的要求；在设计和编程阶段，重点是验证设计和程序实现了需求；而在测试和安装阶段，重点是检查实现的系统符合系统规格说明；在维护阶段，系统将重新测试以决定改变的部分和未改变的部分能继续工作。

表 5-1　生命周期各阶段测试工作划分

生命周期阶段	验证活动
需求	决定验证的方法 决定需求的充分程度 生成功能测试 决定与需求符合的设计
设计	决定设计的充分程度 生成结构和功能测试数据 决定设计与需求的一致性
编程	决定实现的充分程度 生成各种程序/单元的结构和功能测试数据 决定与设计的一致性
测试	决定测试计划的充分性 测试应用系统
安装/集成	把经测试的系统放入产品
维护	修改和重新测试

5.1.2　生命周期测试的主要任务

基于生命周期测试方法对应用系统进行测试的测试工作过程是一个三维过程。一维概述测试要素，二维定义每个阶段要测试的事务，三维是一个测试计划，如图 5-2 所示。测试策略描述测试工程的总体方法和目标。描述目前在进行哪一阶段的测试(单元测试、集成

测试、系统测试)以及每个阶段内正在进行的测试种类(功能测试、性能测试、压力测试等)，给出为什么要执行测试和达到测试目标的最有效途径。这通常由非常熟悉该软件的商业风险的小组开发；而测试种类/技术详细地解释测试的类型或采用的技术，说明执行什么测试和如何进行测试，由测试小组确定测试方法和技术的选择。

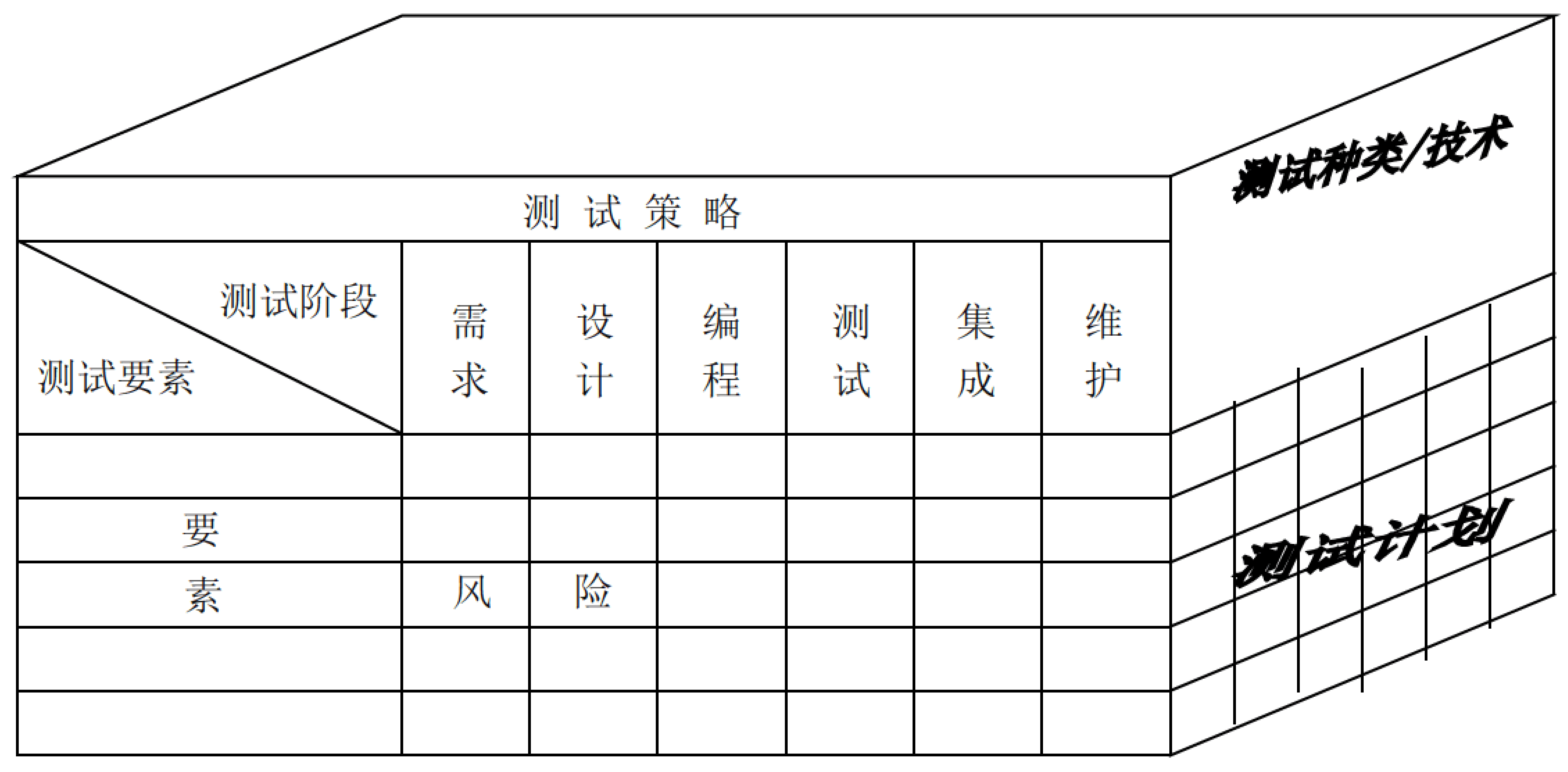


图 5-2　生命周期测试工作三维图

测试要素和测试事件如表 5-2 所示。

表 5-2　测试要素和测试事件

序号	测试要素	需　求	设　计	编　程	测　试	安　装	维　护
1	可靠性	建立精度等级	设计数据完整性控制	实现数据完整性控制	人工、回归和功能测试	验证安装的精度和完整性	修改精度要求
2	授权	定义授权规则	设计授权规则	实现授权规则	符合性测试	禁止改变数据	保存授权规则
	文件完整性	定义文件完整性需求	设计文件完整性控制	实现文件完整性控制	功能测试	检查产品文件的完整性	保存文件完整性
4	审计追踪	定义重构处理需求	设计审计追踪	实现审计追踪	功能测试	记录安装审计追踪	修改审计追踪
5	处理连续性	定义失效的影响	设计中断计划	编写中断计划和过程	恢复性测试	保证以前测试的完整性	修改中断计划
6	服务级别	定义希望的服务级别	设计达到服务级别的方法	达到服务级别的服务系统	强度测试	实现故障安装计划	保存服务级别
7	存取控制	定义系统的存取	设计存取过程	实现安全过程	符合性测试	集成期间的存取控制	保存安全级别
8	方法论	按系统开发方法论定义需求	按系统设计方法论设计系统	按编程方法论编写程序	按测试方法论执行测试	在产品环境中集成系统	按系统维护方法论维护系统

(续表)

序号	测试要素	需　求	设　计	编　程	测　试	安　装	维　护
9	正确性	定义功能规格说明	设计符合需求	程序符合设计	功能测试	程序和数据安装正确	修改需求
10	容易使用	定义可用性规格说明	系统设计要便于实现可用性需求	程序编写符合易用性设计要求，并进行了优化	人工支持测试	传播可用性指令	保存容易使用
11	可维护	决定可维护的规格说明	设计是可维护的	程序是可维护的	检查	文档齐全	保存可维护性
12	可移植	决定可移植的要求	设计是可移植的	程序符合设计	程序符合设计灾难性测试	文档齐全	保存可移植性
13	耦合	定义系统间的接口	设计考虑接口需求	程序符合接口设计说明	功能和回归测试	调整接口	保证正确的接口
14	性能	建立性能准则	保证实际要达到这些准则	程序的实际和实现达到这些准则	符合性测试	监控集成性能	保存性能级别
15	容易操作	定义操作要求	把要求传递给操作	开发操作过程	操作测试	实现操作过程	修改操作过程

我们在确定测试策略时，首先选择并确定测试要素的等级(多数情况下选择 3 至 7 个)，并确定开发阶段；然后明确商业风险，此时开发人员、重要用户和测试人员通过评审的方式对这些风险达成一致的意见；最后把风险列表存放在需求矩阵中，在需求矩阵中可以将风险同测试用例对应起来。

1. 风险

风险是导致失败的条件，计算机系统的风险始终存在。有些风险并不一定导致系统失败。我们不能消除风险，但可以减少风险发生的概率。在软件生命周期各阶段，要标识和评估计算机系统的风险，风险的概念决定测试的类型和做多少测试。决定哪些风险是可以接受的，把这些风险变成测试的领域，然后制定测试计划达到这个目标。

计算机系统的风险表现为：产生不正确的结果、系统接受未授权的事务、破坏计算机文件的完整性、不能重新构造处理、破坏处理的连续性、向用户提供的服务将降低到不可接受的程度、将危及系统的安全、结果不可靠、系统难以使用、程序难以维护、不能移植到其他计算机软硬件环境、不可接受的性能级别以及系统难以操作等。

2. 测试计划

1) 常见问题

在制定测试计划时我们经常会遇到下面的问题：

(1) 测试计划经常是等到开发后期才开始实行，使得没有时间有效地执行计划。

(2) 测试计划的组织者可能缺乏特殊应用软件(如嵌入式软件)的测试经验。

(3) 测试的难度和复杂性可能太大，没有自动化工具，很难计划和控制。

2) 确定测试策略的因素

在确定测试计划时，我们要考虑以下几个方面：

(1) 要使用的测试技术和工具。

(2) 测试完成标准。

(3) 影响资源分配的因素(例如外部接口出现故障、物理设备损坏、安全受到威胁等)。

测试计划最关键的一步就是将软件分解成单元，写成测试需求。测试需求有很多分类方法，最普通的一种就是按照商业功能分类。把软件分解成单元有如下几个好处：

(1) 测试需求是测试设计和开发测试用例的基础，分成单元可以更好地进行设计。

(2) 测试需求是用来衡量测试覆盖率的重要指标。

(3) 测试需求包括各种测试设计和开发以及所需资源。

3) 测试工作量

在测试计划中，估计测试工作量一般要从如下几个方面进行综合考虑：

(1) 效率假设：测试队伍的工作效率。对于功能测试，这主要依赖于应用的复杂度、窗口的个数、每个窗口中的动作数目。对于容量测试，这主要依赖于建立测试所需数据的工作量大小。

(2) 测试假设：为验证一个测试需求所需测试动作数目。

(3) 应用的维数：应用的复杂度指标。例如要加入一条记录，测试需求的维数就是这条记录中域的数详细目。

(4) 所处测试周期的阶段：有些阶段的主要工作是设计，有些阶段的主要工作是测试执行。

(5) 确定测试资源：确定硬件和软件环境以及测试工具等系统资源以及人力资源。其中，最重要的是人力资源，包括测试项目负责人、测试分析员、测试设计员、测试程序员、测试员、测试系统管理者以及配置管理员等。这些工作人员的职责见表 5-3。

表 5-3　软件测试人员配备情况表

工作角色	具体职责
测试项目负责人	管理监督测试项目，提供技术指导，获取适当的资源，制定基线，技术协调，负责项目的安全保密和质量管理
测试分析员	确定测试计划、测试内容、测试方法、测试数据生成方法、测试(软硬件)环境、测试工具，评价测试工作的有效性
测试设计员	设计测试用例、确定测试用例的优先级、建立测试环境
测试程序员	编写测试辅助软件

(续表)

工作角色	具体职责
测试员	执行测试、记录测试结果
测试系统管理员	对测试环境和资产进行管理和维护
配置管理员	设置、管理和维护测试配置管理数据库

注 1：当软件的供方实施测试时，配置管理员由软件开发项目的配置管理员承担；当独立的测试组织实施测试时，应配备测试活动的配置管理员。

注 2：一个人可承担多个角色的工作，一个角色可由多个人承担。

测试计划按国家标准或行业标准规定的格式和内容编写。

3. 测试种类/技术

软件生命周期中所执行的各类测试如图 5-3 所示：

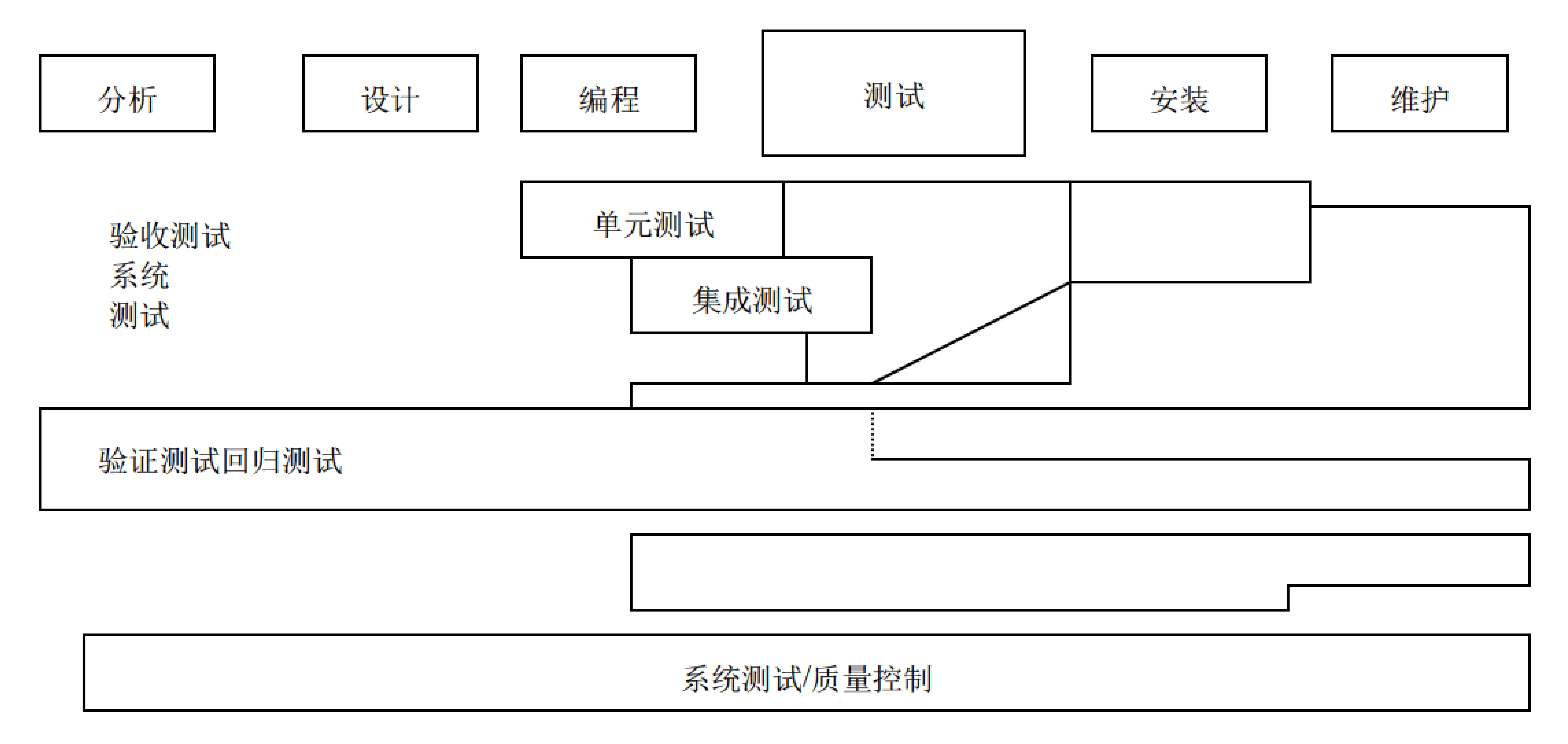


图 5-3　软件生命周期中的各类测试

下面是软件生命周期中各类软件测试的定义和概念：

(1) 质量控制(Quality Control)：

① 决定软件产品正确性的过程和动作；

② 一组功能基线，保证产品符合标准/需求所做的工作。

(2) 缺陷(Defect)：

① 偏离规格说明，有三种表现形式(遗漏、错误和多余)。

② 用户不满意的任何事情，不管是否在规格说明书中规定。

(3) 验证(Verification)：整个软件生命周期中的全部质量控制活动，确保交付的中间产品符合输入规格说明。

(4) 确认(Validation)：软件生命周期中的测试阶段，保证最终产品符合规格说明。

(5) 静态测试：在系统编码之前进行的验证。

(6) 动态测试：在系统编码之后进行的验证和确认。

(7) 单元测试：对单一的独立模块或编码单元进行的测试。

(8) 集成测试：对一组模块进行的测试，确保模块之间的数据和控制能正常地传递。
(9) 系统测试：
① 一个预先确定的测试组合，当执行成功时，系统符合需求(即确认系统开发正确)。
② 与单元测试不同的各种更高等级测试类型的通用术语。
(10) 验收测试：保证系统符合最终用户要求的测试。
(11) 回归测试：在系统改变后进行的测试，以确保不希望的变化不引入系统。
(12) 功能测试：认为系统应该做什么的业务需求测试。
(13) 结构测试：确认系统是如何实现的系统结构测试。
(14) “黑盒”测试：数据驱动的、基于外部规格说明而不用了解系统是如何构造的测试。
(15) “白盒”测试：逻辑驱动的、基于编码内部结构和逻辑的测试。

4. 测试的准入/准出条件

1) 准入条件

进入软件测试生命周期一般应具有下列条件，即我们说的测试准入条件：
(1) 具有测试合同(或项目计划)。
(2) 具有软件测试所需的各种文档。
(3) 所提交的被测软件受控。
(4) 软件源代码正确通过编译或汇编。
(5) 或者能够从一开始就介入被测软件的开发周期。

2) 准出条件

软件测试工作结束周期一般应达到下列要求，即我们说的测试准出条件：
(1) 已按要求完成了合同(或项目计划)所规定的软件测试任务。
(2) 实际测试过程遵循了原定的软件测试计划和软件测试说明。
(3) 客观、详细地记录了软件测试过程和软件测试中发现的所有问题。
(4) 软件测试文档齐全、符合规范。
(5) 软件测试的全过程自始至终在控制下进行。
(6) 软件测试中的问题或异常有合理解释或得到正确有效的处理。
(7) 软件测试工作通过了测试评审。
(8) 全部测试软件、被测软件、测试支持软件和评审结果已纳入配置管理。

5.1.3　基于风险的软件测试方法

风险可以定义为事件、危险、威胁或情况等发生的可能性以及由此产生的不可预料的后果，即一个潜在的问题。

风险级别由出现不确定事件的可能性和出现后所产生的影响(事件引发的不好的结果，即严重性)两个方面来决定。

在软件测试中，由于测试团队需要在时间、成本和质量等各个方面进行平衡和协调，使得我们无法做到穷尽测试，加上受到测试时间以及测试资源的限制(还要考虑测试人员的水平因素)，我们很难达到理想的测试目标：使被测软件“零缺陷”。这样，软件就可能存在缺陷和质量问题，由此导致软件存在着应用上的风险。例如：故障频发的软件交付使用、

软件/硬件对个人或公司造成潜在损害的可能性、劣质的软件特性(如功能性、可靠性、易用性和性能)、低劣的数据完整性和质量(如数据迁移问题、数据转换问题、数据传输问题、违反数据标准问题)、软件没有实现既定的功能等。风险的存在也会导致用户或利益相关者对软件质量或项目成功的信心不足。

1. 什么是基于风险的软件测试

基于风险的软件测试(Risk-Based software Testing，RBT)是指首先评估被测软件的风险，然后根据不同的风险采用不同的测试力度。通常的方法是：

① 列出一个风险的列表；

② 对每个风险进行分析和评估，确定风险级别；

③ 进行考查每项风险的测试；

④ 当风险消失而新的风险出现时，调整测试策略。

现在业界通常采用的对风险进行评估的做法，就是对每个功能点从业务和技术上考查。从业务上是指这项功能失效对系统的影响。从技术上考查是指实现这个功能的技术难度大不大，是移植的还是新研发的？一般将此两项称为重要性和概率，分别赋以 1 到 5 的权值，5 为最大可能或最重要。

对于重要性为 5、概率为 4 的一个功能点，其乘积为 20，这就是一个高的风险。对于高的风险，那么就应该用充足的时间、充足的人员来进行测试。

在基于风险的软件测试中，需要解决的主要问题包括确定测试的优先级、测试重点的有效选择、有效地配置测试资源、分析和评估测试的有效性等。要有效地选择测试重点和测试优先级，风险测试将测试活动和测试任务根据风险划分优先等级，将测试资源主要分配在高风险的部分。

这种基于风险的软件测试方法目前得到了广泛关注，很多机构在他们针对基础级别和进阶级别的测试认证中，将它认定为一种重要的测试方式。

2. 基于风险的软件测试所能解决的问题

基于风险的测试作为软件测试的一种有效方法，可以解决测试过程中面临的一些问题。

1) 测试团队面临的问题是测试任务的时间压力

很少有测试项目可以获得足够的时间进行充分的测试。相反，测试一般都是有时间限制的，例如项目具体里程碑时间、客户或用户要求产品提交的时间等。基于风险的测试可以提供一种方法，对测试用例和测试任务进行优先级排列。测试的时间限制，其面临的挑战实际就是确定测试的覆盖率。通过基于风险的测试，可以从几乎无限的测试中选择重要的和风险高的测试来开展，从而降低风险和尽快提高质量，提高对产品的信心。测试的时间压力不仅仅存在于测试实现和执行阶段，同样也存在于测试分析和设计阶段。通过基于风险的测试，可以在早期将测试工作量放在高风险的地方，同时可以告知利益相关者这样做可能存在的风险。

2) 测试团队经常面临的问题是系统需求质量低下或不完整

通过召集利益相关者讨论哪些是需要测试的、哪些是不需要测试的、测试的深度是多少等问题，基于风险的测试可用来识别需求规格说明中存在的不足。基于风险的测试也可

以帮助其他利益相关者认识到测试在确定测试范围和测试深度方面面临的挑战，有助于项目团队成员之间更好地理解和沟通。

3) 在项目测试的后期，例如完成测试执行之后，测试团队需要提供相关的信息给其他的利益相关者，以帮助做出合适的决定

基于风险的测试可以允许测试团队和其他利益相关者一起，根据剩余的风险确定一个可接受的风险级别，而不是仅仅依赖于其他一些不充分的度量，例如缺陷数目、测试用例执行数目等。

3. 基于风险的软件测试活动实践

有效地应用基于风险的软件测试可以较好地指导软件测试活动的展开，更好地使软件测试活动在时间、成本、质量等方面进行平衡，从而提高测试质量、降低测试成本、缩短测试时间等。下面是基于风险的软件测试方法。

1) 确定测试优先级

根据测试风险的分析和评估得到的风险分布，确定测试的优先级(风险级别分析也适用于测试的设计和测试实现等阶段，即通过风险分析，确定测试设计和测试实现的优先级)。测试风险的分析基于两个方面：发生的可能性和发生的严重程度。其中，风险发生的可能性主要是从技术方面考虑；而风险发生的严重程度主要是从客户或用户的角度考虑。

2) 确定测试完备性

前面提到一个假设条件：并不是所有的测试对项目而言都是同等重要的。同样的道理，并不需要对测试对象的不同内容进行同等重要的测试，例如：最重要或风险最大的模块或对象需要测试得更加彻底、更加完备；而对于风险比较小、优先级低的模块或对象，可以简单测试。对于优先级最低的对象，在时间和成本等不允许的时候，甚至不进行测试。

3) 确定测试资源分配

根据测试风险的分析和测试优先级的评估，将经验丰富和技术能力丰富的测试人员(不管是设计人员、实现人员还是执行人员)放在最重要的模块或测试对象中，以达到最佳效果：①设计更加完善、完备和准确的测试用例；②实现高质量的测试用例脚本和代码；③更加高效地发现测试对象中的缺陷。

4) 监控测试进度

根据测试风险的分析和评估，得到测试的优先级和测试重点。接下来，可以根据风险的分布对测试进度进行汇报和控制。例如：测试经理可以根据测试工作的侧重点、测试进度协调人力资源和测试环境的分配，将测试资源放在最重要的部分。

5) 加速测试信心提升

依据测试风险分析和评估得到的测试优先级和测试重点，可以更好、更快地提供产品或被测系统在质量方面的信心。对被测对象的质量，根据不同的测试策略，得到不同的信心演变过程。

策略 1：随机执行测试用例，不分优先级和测试重点，被测系统质量信心的递增是随着测试完成率的递增而线性增加的。

策略 2：先执行复杂度低的测试，因此，测试完成率增加很快，但是相应的被测对象质量的信心却增加很慢；而对于高风险(例如测试难度较大的大容量用户数据模拟测试)的区域，很可能放在测试的后期进行。

策略 3：基于风险的测试，对高风险区域首先进行测试，尽管测试完成率增加比较慢，但是对被测对象质量的信心却增加很快。

5.2　生命周期各个阶段的测试要求

全生命周期中软件测试的最终要求是：①保证软件系统在全生命周期中每个阶段的正确性，验证在整个软件开发周期中各个阶段的软件质量是否合格；②保证最终系统符合用户的要求和需求，验证最终交付给用户的系统是否满足用户需要、符合其需求；③用样本测试数据检查系统的行为特性；④测试的最终目的是确保最终交给用户的产品的功能符合用户的需求，原则是把尽可能多的问题在产品交给用户之前发现并改正。为此，我们要努力保证生命周期中每个阶段的正确性，使其满足阶段出口的要求。

5.2.1　需求阶段测试

据软件工程统计结果，发现 50%以上的系统错误是由于错误需求或缺少需求导致的，在需求上发生错误将导致相互纠缠和重复劳动，因而测试费用的 80%花在对需求错误的追踪上。

需求测试贯穿整个软件开发周期，通过需求测试可以知道软件测试的各个阶段，帮助我们设计整个测试的进行、测试计划的安排、测试用例的设计以及软件的确认要达到哪些要求等。有了正确的需求分析，大部分缺陷将不会进入到设计和编码阶段，测试所需的费用自然会大大减少，因此，需求阶段测试的所有花费都是值得的。

1. 需求阶段测试的目标

简单来说，需求阶段测试的目标就是保证需求分析的正确性和充分性。具体地说，需求阶段测试的目标则是保证需求正确表现出了用户的需要，需求已经被定义和文档化，项目的花费和收益成正比，需求的控制被明确，有合理的流程可以遵循，有合理的方法可供选择。

2. 需求阶段的测试要素分析

需求阶段的测试要素分析以表 5-2 为基础，包含如下内容：

① 需求的设计是否遵循已定义的方法；
② 提交了已定义的功能说明；
③ 定义了系统界面；
④ 已经估计了性能标准；
⑤ 容忍度被预先估计；
⑥ 预先定义了权限规则；

⑦ 需求中预先定义了文件完整性；
⑧ 预先定义了需求的变更流程；
⑨ 预先定义了失败的影响。

3. 需求阶段的测试活动

在需求阶段测试中，需要建立风险列表，进行风险分析和检查，以此确定项目的风险；并且要建立控制目标，确保有足够的控制力度来保证软件项目的开发和测试。在彻底地分析需求的充分性后，生成基础的测试用例。澄清和确定哪些需求是可测试的，舍去含糊的、不可测试的需求，建立产品的需求和确认需求。

5.2.2　设计阶段测试

在设计阶段，设计人员需要根据需求分析详细定义要交付的产品——硬件和软件的需求、操作手册说明书、数据保留的策略、输入/输出说明、过程说明、控制说明、系统流程图等。而测试的任务是对设计进行评审，分析测试要素，给测试要素打分，当需求分析发生改变，设计文档也要修改时，测试要对修改的部分进行检查，以保证设计和需求的一致性。

1. 设计阶段的测试活动

设计阶段包括概要设计和详细设计。在概要设计阶段，测试人员应阐述测试方法和测试评估准则，编写测试计划，组织成立一个独立的测试小组，安排具有里程碑的测试日程；在详细设计阶段，测试人员要开发或获取确认支持工具，生成功能测试数据和测试用例，以此来检查设计中遗漏的情况、错误的逻辑、模块接口的不匹配、数据结构不合理、错误的 I/O 假定、用户界面的不充分等。

2. 设计阶段的评审

设计阶段的评审是对实际阶段处理的完整性进行正式的评价。在对设计进行评审之前，要为评审分配足够的时间，成立评审组，并对组员进行培训；在评审时，要通报项目组，和项目组一起进行评审，并且只对文档进行评审；最后，要将评审的结果写成正式报告。

3. 设计阶段工具的应用

在设计阶段使用静态和动态测试工具测试系统的结构。评分工具和设计评审工具是广泛使用的两种测试工具，评分是标识风险的一种工具，根据得分的结果确定系统的风险程度；而设计评审是对实际阶段处理的完整性进行正式的评价，是测试设计规格说明的工具，风险越高，设计评审越详细。

另外，我们可利用评分工具对测试要素进行分析，给测试要素打分，例如：是否设计了对数据完整性的控制？是否设计了权限规则？是否设计了对文件完整性的控制？是否设计了审计追踪？是否设计了发生意外情况时的计划？是否设计了如何达到服务水平的方法？是否定义了权限流程？是否定义了完整的方法学？是否设计了保证需求一致性的方法？是否进行了易用性的设计？设计是否可维护、简单？交互界面设计是否完毕？是否定义了成功的标准？

上述评分过程需要同实际操作者沟通。

5.2.3　编码阶段测试

在编码阶段，测试需要解决的首要问题是编码是否和设计一致；其次是系统是否可维护，系统的规格说明是否正确地实现，编码是否按照既有的标准进行，是否有充分的测试计划来评价可执行的程序，程序是否提供了足够的文档资料，程序内部是否有足够的注释等。在测试完成后，要形成下列输出：编码说明书、程序文档、计算机程序列表、可执行的程序、程序流程图、操作介绍和单元测试结果。

在编码阶段已经开发了很多的测试工具，例如支持程序走查和检查的代码静态分析工具和支持单元“黑盒测试”和单元“白盒测试”的动态测试工具。在编码阶段的测试活动中，有几个方面是需要特别关注的：

① 完成对数据和文件完整性的控制；
② 定义完毕授权的规则；
③ 实现审计追踪；
④ 规划出意外情况发生后的处理计划；
⑤ 编码工作是依据规定的方法完成的，这样易于测试和维护工作的进行；
⑥ 编码与设计相一致，包括编码的正确性、易用性、间接性和耦合性；
⑦ 代码是可维护的。代码的维护性在一定程度上决定了项目维护的难易程度；
⑧ 在性能上定义程序成功的标准。

5.2.4　测试阶段

测试阶段就是传统软件工程中的软件测试。在全生命周期软件测试方法中，由于在需求、设计、编码阶段都进行了测试，因此测试阶段的问题相对传统的软件测试中的问题要少一些。在测试阶段要进行第三方的正式确认测试，检验所开发的系统是否能按照用户提出的要求运行。在测试阶段要使得用户能成功地安装一个新的应用系统来进行测试。

1. 典型的测试类型

典型的测试类型如下：

1) 手册与文档测试

测试软件的操作说明文档是否全面、正确、简单和满足标准，即测试软件文档的易用性。

2) 一致性测试

测试软件的授权、安全性和性能是否达到需求分析中的要求。

3) 符合性测试

验证软件系统与相应标准的符合程度，如授权规则是否正确实现，安全方法是否合适，是否按照相应的标准、指南、规程执行测试等。

4) 功能测试

运行部分或全部系统，确认用户的需求被满足，包括对可靠性、文件完整性、审计追踪、功能正确性、互连等的测试，检验系统在各种环境和重复的事务条件下能否正确地执行系统的需求，控制计算机文件的完整性，追踪一个原始事务到总的控制，按用户规定的

需求测试应用功能，与其他应用系统能否正确地通信，等等。

5) 覆盖性测试

检验软件代码各个语句及分支等是否全部执行到。

6) 性能测试

通过测量响应时间、CPU 使用和其他量化的操作特征，评估软件系统的性能指标。

7) 压力测试

以大信息量的数据进行输入，测试软件的性能。但这是一项很昂贵的测试，应根据需要来选择。但是在线系统必须进行压力测试。

8) 强度测试

将系统置于强度下进行验收测试，测试系统对极端条件的反应，标识软件的薄弱点，指出系统能够经受的正常的工作量。

9) 操作测试

在没有开发人员的指导和帮助的情况下，由操作人员进行测试，以评估操作命令的完整性和系统是否容易操作。

10) 恢复测试

故意使系统失败，测试人工和自动的恢复过程。

2. 测试用例

设计测试用例的输入数据时，要包含合法和非法的输入(要尝试让测试数据违反程序的规则并进行输入)，也要描述执行测试用例的预期结果。测试用例应包含的一些基本信息有标识符(测试设计过程说明和测试程序说明引用的唯一标识符)、输入说明(列举软件执行测试用例的所有输入内容或条件)、输出要求(描述执行测试用例的预期结果)、环境要求(执行测试用例必要的硬件、软件等)、特殊过程要求(描述执行测试必须达到的特殊要求)以及用例之间的依赖性(当一个测试用例依赖于其他测试用例，或者受其他测试用例影响时，需要在此说明)。

3. 测试报告

在测试阶段开始之前，测试人员要参考前期的测试结果和第三方测试反馈(例如计算机操作人员)准备测试阶段的测试计划和测试用例，在完成测试时要生成正式的测试总结报告。生成测试报告的目的是要表示出目前项目的实际情况，给出系统的操作性评价，为软件的产品化提供参考。

测试总结报告的内容有测试结果数据；测试任务、测试集合和测试事件的描述；目前的软件状态描述；各个阶段的项目测试总结等。

5.2.5　安装阶段测试

在进行安装测试时要保证被测系统没有问题，校验产品文件的完整性，安装要遵循一定的方法和步骤；注重对程序安装的正确性和完整性进行核对；如果安装失败，系统要有

相应的解决方案；最后也是最重要的是要保证系统综合的性能达到了用户要求。

1. 安装测试的基本要求

在安装阶段，我们首先要根据系统安装手册制定好安装计划，确定好安装流程图，准备好安装文件和程序清单，给出安装测试的预期结果，并对安装过程中的各项可能发生的结果进行说明准备，将程序运行的软硬件要求放入产品说明中。同时要检查系统的用户手册和操作手册，看是否可用。

2. 安装测试工作

安装过程中我们要进行如下工作：

(1) 对程序安装的正确性和完整性进行核对。
(2) 校验产品文件的完整性。
(3) 对安装审查，追踪被记录。
(4) 安装之前，该系统已经被证实没有问题。
(5) 如果安装失败，系统有相应的解决方案。
(6) 对安装过程进行了权限控制(安全性)。
(7) 安装遵循一定的方法、步骤。
(8) 需要的配套程序和数据已经放进产品中。
(9) 已交付使用说明。
(10) 相关文件已经完整(可维护性)。
(11) 接口已经被合理调整(耦合性)。
(12) 综合性能达到了用户要求。

5.2.6　验收阶段测试

软件验收的流程如下：

1. 定义用户角色

定义用户角色也就是确定软件的用户范围。

2. 定义验收标准

验收标准包括功能、性能、接口质量、软件的安全性和稳定性等方面的标准。

3. 编制验收计划

验收计划包含项目描述、用户职责描述、验收活动描述、验收项的评审和最终的验收测试步骤。

4. 执行验收计划

按照验收计划进行测试和评审。

5. 填写验收结论

验收结束后，填写得出的结论，验收问题必须在进入下一个活动之前被接受和更改。

5.2.7　维护阶段

软件交付使用后的阶段，称为维护阶段。

软件维护阶段的工作重点是测试和培训。

1. 维护阶段中的测试要求

由于软件产品的特殊性，测试过后，并不代表软件没有错误，只能说有些错误还没有被发现，因此在软件交付使用后，仍旧需要对其进行维护。维护人员需要开发一些测试用例，预先发现一些问题；并且要能够根据运行情况的变化和用户的反馈对软件做适当的修正。

2. 维护阶段中的培训要求

在软件交付使用的同时，也要制定培训计划，编写培训材料。培训计划包括对系统进行概览，对系统假定的一些错误给出处理的方法；培训材料包括用户使用方法、对错误列表中的问题给出解释、对输入数据进行解释等。

5.3　生命周期软件测试案例分析

5.3.1　被测样例系统需求说明

1. 引言

1) 目的

被测样例系统是高校学生选课系统，本系统在能满足所要求的安全需求的同时，采用简洁的操作对高校学生选课系统的运行环境、基本功能、业务流程、使用方法等进行阐述，为软件设计方案提供指导依据。

本需求说明适用于被测样例系统。

2) 系统概述

(1) 概述

高校学生选课系统一般应用于学生用户和管理员用户，要求系统界面设计美观大方、操作简捷灵活；在该系统中学生用户必须实现课程信息的具体操作及管理，包括用户注册、课程选择、查看历史选课情况、已选课程信息统计等功能。

(2) 系统功能

本系统将用户角色分为两种，分别为学生模块和管理模块。学生可以选择课程，管理员可以添加、修改、删除课程。通过登录功能可以完成管理员用户登录和学生用户登录。登录成功进入系统后，可以实现对专业、课程、统计信息、用户密码的操作。

(3) 设计约束

① 开发环境：Windows 7 操作系统；

② 编译工具：Eclipse 运行环境；

③ 数据库：MySQL。

2. 功能性需求

1) 专业管理

专业管理模块可分为面向管理员和面向学生两个方向。管理员可通过单击“专业管理按钮对所有专业进行查看，添加新专业，对各个专业的结业情况进行管理；学生可通过单击“专业管理”按钮对专业下所有课程的开设年份、专业学制、结业情况等进行查看。

2) 课程管理

管理员可通过单击“课程管理”按钮对所有课程及其具体信息进行查看，也可添加新课程，并设置课程面向学生是否可选；学生可通过单击“课程管理”按钮查询课程具体信息，如专业信息、课程信息、授课教师信息，也可选课。

3) 统计信息

管理员和学生均可通过单击“统计信息”按钮查看专业、课程的具体信息，管理员可以通过此功能查看某课程对应的学员名单。

4) 修改密码

管理员和学生均可单击“修改密码”按钮，通过验证自己的用户名、旧密码、验证邮箱来设置自己的新密码并使用。

5) 退出系统

管理员和学生均可通过单击“退出系统”来注销账号并回到登录界面，以便下次使用。

3. 非功能性需求

1) 性能需求

根据学生和管理员对本系统的要求，确定系统在响应时间、系统可靠性、系统安全性等方面有较高的性能要求。

2) 界面需求

系统的界面要求如下：

(1) 页面内容：主题突出，站点定义、术语和行文格式统一、规范、明确，栏目、菜单设置和布局合理，传递的信息准确、及时；内容丰富，文字准确，语句通顺；专用术语规范，行文格式统一、规范。

(2) 导航结构：页面具有明确的导航指示，且便于理解，方便用户使用。

(3) 技术环境：页面大小适当，能用各种常用浏览器以不同分辨率浏览；无错误链接和空链接；采用 CSS 处理，控制字体大小和版面布局。

(4) 艺术风格：界面、版面形象清新悦目、布局合理，字号适宜、字体选择合理，前后一致，美观大方；动与静搭配恰当，动静效果好；色彩和谐自然，与主题内容相协调。

3) 响应时间需求

无论是学生端还是管理端，当用户登录进行任何操作的时候，系统应该及时地进行响应，响应的时间在 5 秒以内。系统应能监测出各种非正常情况，如与设备的通信中断、无

法连接数据库服务器等，避免出现长时间等待甚至无响应。

4) 可靠性需求

系统应保证 7×24 小时不宕机，保证 100 人可以同时在客户端登录，系统正常运行，正确提示相关内容。

5) 开放性需求

系统应具有灵活性，以适应将来功能扩展的需求。

6) 可扩展性需求

系统设计要求能够体现扩展性要求，以适应将来功能扩展方面的需求。

7) 系统安全性需求

系统有严格的权限管理功能，各功能模块必须有相应的权限方能进入。系统必须能够防止因为各种误操作可能造成的数据丢失、破坏。防止用户非法获取网页以及内容。

8) 软件接口需求

通过高校学生选课系统的主界面可以选择四个主功能，由四个功能的任意一个按钮可以进入专业管理模块、课程管理模块、统计信息模块和密码修改模块并进行相应操作。

5.3.4 被测样例系统设计说明

1. 目的

本设计说明的编写目的是说明程序模块的设计考虑，包括程序全局数据结构说明、功能描述、输入/输出数据等，为软件编程和系统维护提供基础，为后续软件开发和测试提供可靠依据。

2. 全局数据说明

下面主要以功能模块为单位介绍各模块重要的常量、变量等情况，包括数据含义和数值信息，如表 5-4~表 5-8 所示。

表 5-4　专业管理功能全局数据

变量名称	含义
enterYear	开设专业的年份
name	专业的名称
langthYear	专业的学年
isFinish	是否增加按钮

表 5-5　课程管理功能全局数据

变量名称	含义
id	课程编号，自动增长主键
name	课程名称
schooltime	上课时间
addr	上课地点

(续表)

变量名称	含义
credit	课程学分
courseInfo	课程介绍
teacherName	讲课老师姓名
teacherInfo	讲课老师介绍
isFinish	课程是否可选(1 为可选，0 为不可选)
specialtyId	所属专业编号

表 5-6　统计信息功能全局数据

变量名称	含义
id	学生编号
stuName	学生姓名
stuNo	学生学号
specialtyId	学生所学专业编号
stuSex	学生性别
birthday	学生出生年月日
homeAddr	学生家庭地址
addr	学生当前居住地址

表 5-7　修改密码功能全局数据

变量名称	含义
id	学生编号
loginName	登录用户名
pwd	登录旧密码
type	更改新密码
mail	验证邮箱

表 5-8　登录功能全局数据

变量名称	含义
id	学生编号
loginName	登录用户名
pwd	登录旧密码
type	更改新密码
mail	验证邮箱

3. 功能设计说明

1) 专业管理功能

在专业管理功能里，学生可以查看所有专业，管理员可以单击增加新专业，填写新专业信息来开设新的专业，也可以对专业的结业情况进行操作。

下面选取实现专业管理功能中的增加新专业的 SpecialtyAction.class 中的 insert 函数，进行详细的设计说明，具体如下：

(1) 程序描述

本函数可根据填写的新专业信息(入学年份、专业学分、学制)来增加相应的新专业。

(2) 功能

管理员添加新专业并完善专业信息。

(3) 输入数据

- 入学年份(int enterYear)
- 专业名称(String name)
- 学制(int langthYear)

(4) 输出数据

新专业出现在专业清单列表中。

2) 课程管理功能

在课程管理功能里，学生可以查看各个专业所开设的课程信息；管理员可以单击增加新课程，填写新课程信息，开设新的课程。

下面选取实现课程管理功能的主要函数进行详细的设计说明，具体如下：

(1) CourseAction 中的函数 insert。

① 程序描述

通过本函数，管理员可以通过填写新课程信息(如选择专业、课程名称、上课时间、上课地点、课程学分、课程介绍、授课教师、教师介绍)来添加新课程，用以增加新课程及课程内容。

② 功能

管理员添加新课程并完善课程信息。

③ 输入数据

- 选择专业(int specialtyId)
- 课程名称(String name)
- 上课时间(String schooltime)
- 上课地点(String addr)
- 课程学分(int credit)
- 课程介绍(String courseInfo)
- 授课教师(String teacherName)
- 教师介绍(String teacherInfo)

④ 输出数据

新专业课程清单及其信息。

(2) CourseAction 中的函数 findBySearch

① 程序描述

通过本函数，学生可以搜索新课程，并通过单击新课程名称来查看具体信息，如选择专业、课程名称、上课时间、上课地点、课程学分、课程介绍、授课教师、教师介绍，根

据信息进行选课。

② 功能

学生可查询课程并选课。

③ 输入数据

- 选择专业(int specialtyId)
- 课程名称(String name)
- 授课教师名称(String teacherName)

④ 输出数据

学生所查询的课程信息，选课结果。

3) 统计信息功能

在统计信息功能里，管理员可以查看各个课程的学员名单及信息；学生可以查找自己的选课信息。

下面选取实现统计信息功能的 StatInfoAction.class 中的函数 findBySearch，进行详细的设计说明，具体如下：

(1) 程序描述

管理员可以查找课程，并查看某课程对应的学员名单；学生可以查看自己的选课信息。

(2) 功能

管理员查询课程学员名单信息。

(3) 输入数据

- 选择专业(int specialtyId)
- 课程名称(String name)
- 授课教师名称(String teacherName)

(4) 输出数据

某课程对应的学员清单列表。

4) 修改密码功能

修改密码功能可以实现对学生和管理员旧密码的修改。

下面选取实现修改密码功能的 UpdataPwdAction 中的函数 execute，进行详细的设计说明，具体如下：

(1) 程序描述

通过本函数，学生和管理员可以通过验证自己的身份，对现有的密码进行修改。

(2) 功能

管理员和学生修改密码功能。

(3) 输入数据

- 登录账号(String loginName)
- 旧密码(String pwd)
- 新密码(String pwd1)
- 确认密码(String pwd1)
- 验证邮箱(String mail)

(4) 输出数据

“密码修改成功”字样显示在“用户名”文本框的上方。

5) 登录验证功能

登录验证功能可以实现对学生和管理员进入选课管理系统的基本身份验证，保障系统的安全性。

下面选取实现修改密码功能的 UserLoginAction 中的函数 execute，进行详细的设计说明，具体如下：

(1) 程序描述

通过本函数，学生和管理员可以验证自己的身份并进入系统。

(2) 功能

登录验证功能。

(3) 输入数据

- 登录账号(String loginName)
- 密码(String pwd)

(4) 输出数据

自动进入选课管理系统。

习题和思考题

1. 详细说明软件工程生命周期 V 形图的含义。
2. 怎样才能把好软件工程生命周期各个阶段的质量关？
3. 什么是生命周期测试方法？生命周期测试如何开展？
4. 生命周期测试有哪些测试任务？简述测试策略、测试要素及测试风险各自的含义。
5. 计算机系统的风险表现举例？简述基于风险的软件测试方法。
6. 如何制定测试计划？在制定测试计划时，应考虑哪些因素？
7. 测试阶段有哪些测试内容？用到哪些技术？
8. 在需求阶段、设计阶段、编码阶段、测试阶段、安装阶段、验收阶段及维护阶段需要进行哪些测试？

第6章　软件测试过程及测试过程管理

通过对前面章节的学习，你了解到软件测试不能等到编码完成后才开始对程序进行测试，而是在项目需求分析阶段就要参与进去，审查需求分析文档、产品规格说明书；然后在设计阶段，审查系统设计文档、程序设计流程图、数据流图等；在代码测试阶段，需要审查代码，看是否遵循守代码的变量定义规则、是否有足够的注释行等。因此，从软件开发生命周期角度看，软件测试也存在着生命周期概念，即软件测试生命周期。软件测试生命周期一般包括 7 个阶段：计划、分析、设计、构建、测试用例编写、最后测试实施以及测试总结。测试的生命周期是软件生命周期的一部分，应该与开发周期同时开始，否则会导致在开发时间表上附加一个长时间、高成本的测试和错误修正时间表。

软件测试过程与软件开发一样，可用一种抽象模型表示，用于定义软件测试的流程和方法，指导测试团队按照承担软件测试项目所要求的进度、成本和质量开展测试任务，必须覆盖整个软件测试生命周期的一组有序的软件测试活动。众所周知，软件开发过程的质量决定了软件的质量，同样，软件测试过程的质量将直接影响测试结果的准确性和有效性。软件测试过程和软件开发过程一样，都遵循软件工程原理及管理学原理。因此，一般意义上的软件测试过程包括：软件测试需求分析、测试计划制定、测试用例设计、测试工具开发、测试环境构建、测试实施及测试评估等阶段，每个阶段都有一系列的任务。

此时，软件测试不再只是对程序代码执行测试工作，而是包含了所有在软件开发过程阶段产出物的测试工作，这扩展了测试工作的范围，更有利于发现和控制软件开发中的缺陷。

另外，软件开发与软件测试应该是交互进行的，例如，单元编码需要单元测试，模块组合阶段需要集成测试。如果等到软件编码结束后才进行测试，那么测试的时间将会很短，测试的覆盖面将不全面，测试的效果也将打折扣。更严重的是，如果此时发现软件需求阶段或概要设计阶段的错误，要修复该类错误，将会耗费大量的时间和人力。

而且，软件测试过程清晰地定义了过程的输入/输出流，为当前整个测试过程实施的度量和今后测试过程的改进奠定了基础。

最后，要强调将测试的结果纳入软件开发的缺陷管理机制，扩展测试的作用。

6.1　软件测试过程

软件测试过程要求基于项目的整体需求，对整个测试生命周期中的所有过程、活动及变更进行定义、控制和管理。在当前基于软件生命周期的开发过程中，人们经常用到的主

流软件生命周期模型或软件开发过程模型有瀑布模型、原型模型、螺旋模型、增量模型、渐进模型、快速软件开发(RAD)以及 Rational 统一过程(RUP)等，这些模型对于软件开发过程具有很好的指导作用。但是在这些过程方法中，软件测试的地位和价值并没有体现出来，也没有给软件测试以足够的重视，利用这些模型无法更好地指导测试实践。软件测试是与软件开发紧密相关的一系列有计划、系统性的活动，显然软件测试也需要测试模型来指导实践。

6.1.1　软件测试过程模型

对软件测试过程模型的研究随着软件工程的发展而越来越深入，在 20 世纪 80 年代后期 Paul Rook 提出了著名的软件测试的 V 模型，旨在改进软件开发的效率和效果。

1. V 模型

在传统的开发模型中，比如瀑布模型，通常把测试过程作为在需求分析、概要设计、详细设计和编码全部完成之后的一个阶段，尽管有时测试工作会占用整个项目周期一半的时间，但是有人仍认为测试只是一项收尾工作，而不是主要的工程。V 模型是软件开发瀑布模型的变种，反映了测试活动与分析和设计的关系，从左到右，描述了基本的开发过程和测试行为，明确地表明了测试过程中存在不同类型、不同级别的测试，清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系，如图 6-1 所示。

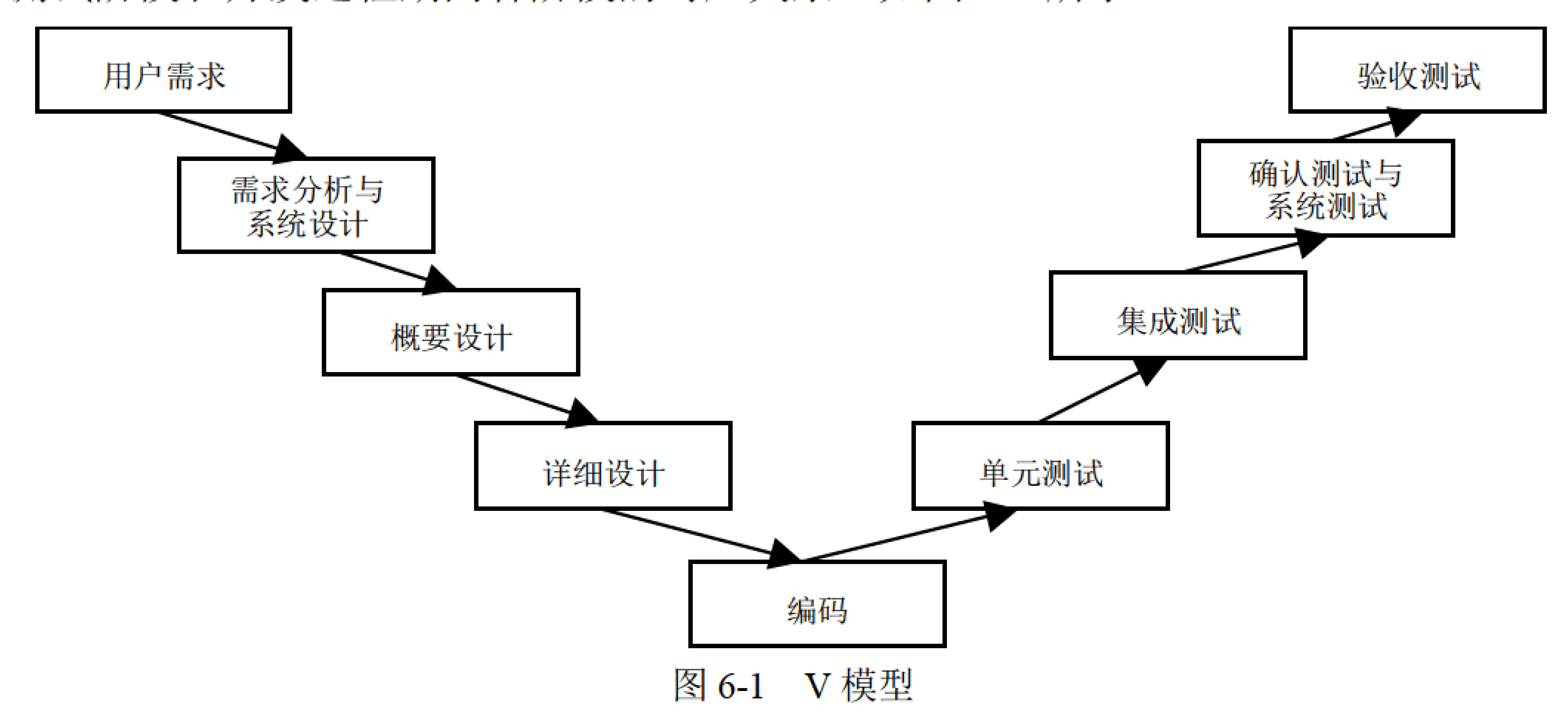


图 6-1　V 模型

在图 6-1 中，V 模型中的箭头代表了时间方向，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。

V 模型的软件测试策略既包括低层测试，又包括高层测试，低层测试是为了源代码的正确性，高层测试是为了使整个系统满足用户的需求。

V 模型指出，单元和集成测试是验证程序设计，开发人员和测试组应检测程序的执行是否满足软件设计的要求；系统测试应当验证系统设计，检测系统功能、性能的质量特性是否达到系统设计的指标；由测试人员和用户进行软件的确认测试和验收测试，追溯软件需求说明书进行测试，以确定软件的实现是否满足用户需求或合同的要求。

V 模型存在一定的局限性，它仅仅把测试过程作为需求分析、概要设计、详细设计及编码之后的一个阶段。容易使人理解为测试是软件开发的最后一个阶段，主要针对程序进

行测试寻找错误，而需求分析阶段隐藏的问题一直到后期的验收测试才被发现。

2. W 模型

V 模型的局限性在于没有明确地说明早期的测试，不能体现“尽早和不断地进行软件测试”的原则。在 V 模型中增加软件各开发阶段应同步进行的测试，被演化为一种 W 模型，如图 6-2 所示。

相对于 V 模型，W 模型更科学。W 模型可以说是 V 模型自然而然的发展。它强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要测试。这样，只要相应的开发活动完成，我们就可以开始执行测试。可以说，测试与开发是同步进行的，从而有利于尽早地发现问题。以需求为例，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，以减少总体测试时间，加快项目进度。

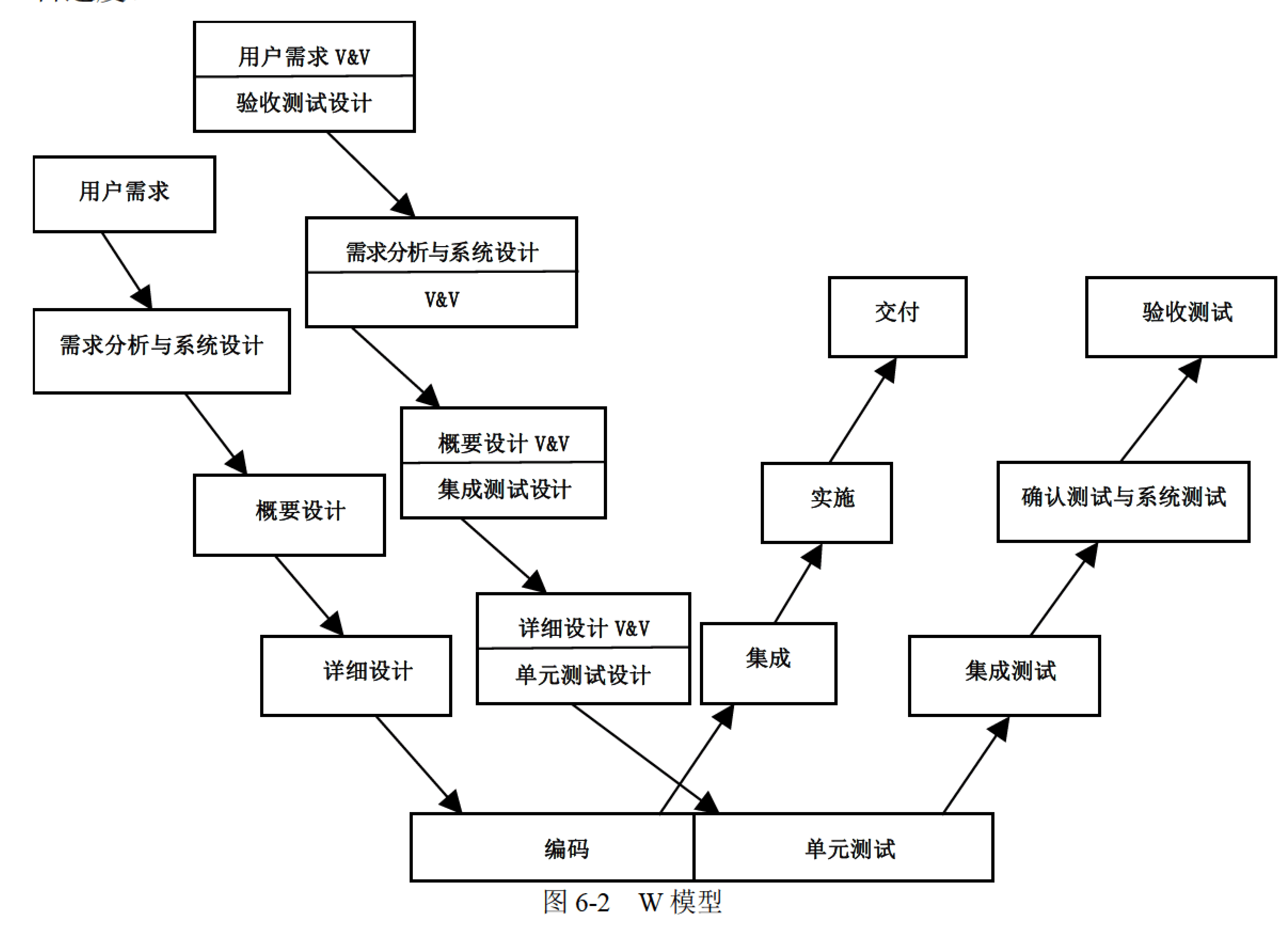


图 6-2　W 模型

如果测试文档能尽早提交，那就有了更多的检查和检阅时间，这些文档还可用于评估开发文档。另外还有一个很大的好处，就是测试者可以在项目中尽可能早地面对规格说明书中的挑战。这意味着测试不仅仅是评定软件的质量，测试还可以尽可能早地找出缺陷所在，从而帮助改进项目内部的质量。参与前期工作的测试者可以预先估计问题和难度，这将可以显著地减少总体测试时间，加快项目进度。

根据 W 模型的要求，一旦有文档提供，就要及时确定测试条件，以及编写测试用例，这些工作对测试的各级别都有意义。当需求被提交后，就需要确定高级别的测试用例来测

试这些需求。当概要设计编写完成后，就需要确定测试条件来查找该阶段的设计缺陷。

W 模型也是有局限性的。W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列串行的活动。同样，软件开发和测试保持一种线性的前后关系，需要有严格的指令来表示上一阶段完全结束，才可以正式开始下一个阶段。这样就无法支持迭代、自发性以及变更调整。对于当前很多文档需要事后补充，或者根本没有文档的情况(这已成为一种开发文化)，开发人员和测试人员都面临同样的困惑。

3. H 模型

V 模型和 W 模型均存在一些不妥之处。首先，如前所述，它们都把软件开发视为需求、设计、编码等一系列串行的活动，而事实上，虽然这些活动之间存在相互牵制的关系，但在大部分时间内，它们是可以交叉进行的。虽然软件开发期望有清晰的需求、设计和编码阶段，但实践告诉我们，严格的阶段划分只是一种理想状况。特别是现在，很少有软件项目是在有了明确的需求之后才开始设计的。所以，相应的测试之间也不存在严格的次序关系。同时，各层次之间的测试也存在反复触发、迭代和增量关系。其次，V 模型和 W 模型都没有很好地体现测试流程的完整性。

为了解决以上问题，提出了 H 模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来。H 模型如图 6-3 所示。

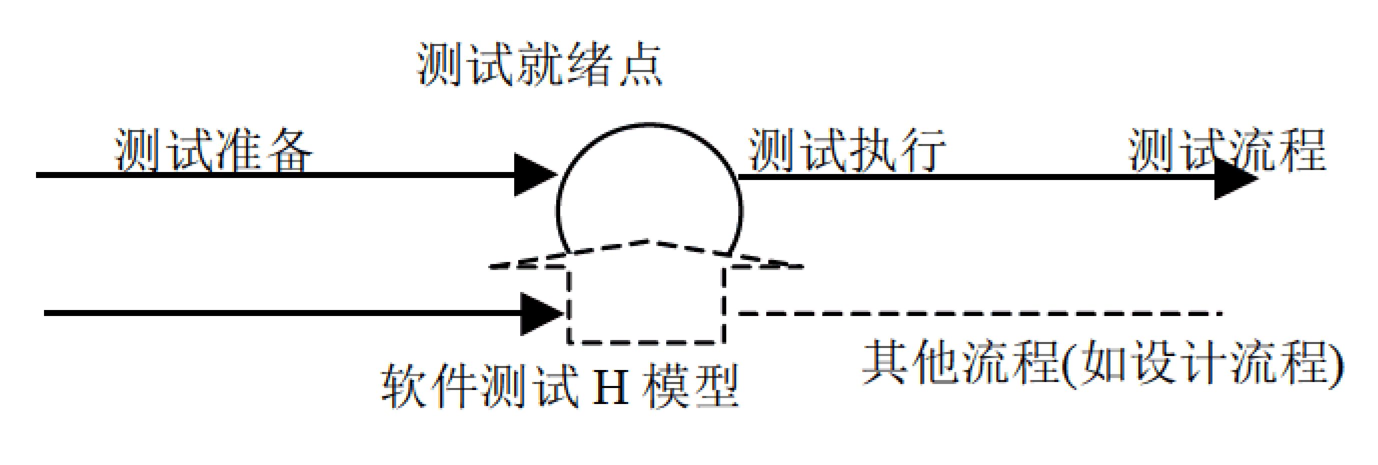


图 6-3　H 模型

H 模型仅仅演示了在整个生产周期中某个层次上的一次测试“微循环。图 6-3 中的其他流程可以是任意开发流程，例如设计流程和编码流程。也可以是其他非开发流程，例如 SQA 流程，甚至是测试流程自身。也就是说，只要测试条件成熟，测试准备活动完成了，测试执行活动就可以(或者说需要)进行了。

概括地说，H 模型揭示了：①软件测试不仅仅指测试的执行，还包括很多其他的活动；②软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行；③软件测试要尽早准备，尽早执行；④软件测试是根据被测物的不同而分层次进行的。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复进行的。

在 H 模型中，软件测试模型是一个独立的流程，贯穿于整个产品周期，与其他流程并发地进行。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

4. 其他测试模型

除了上述三种常见模型外，还有其他几种模型，如 X 模型、前置测试模型等。X 模型提出针对单独的程序片段进行相互分离的编码和测试，然后通过频繁的交接，通过集成最终合成为可执行程序。这些可执行程序还需要进行测试。已通过集成测试的部件可以进行打包并提交给用户，也可以作为更大规模和范围内集成的一部分。

前置测试模型体现了开发与测试的结合，要求对每个交付的内容进行测试。这些模型都针对其他模型的缺点进行了一些修正，但本身仍然存在一些不足的地方。所以在软件测试过程中，正确选取模型是一个很关键的问题。

5. 测试模型的使用

前面介绍的几种典型的测试模型，应该说它们对指导测试工作很有意义，但任何模型都不是完美的。我们应该尽可能地应用模型中对项目有实用价值的方面，但不强行地为使用模型而使用模型，否则也没有实际意义。

在这些模型中，V 模型强调了在整个软件项目开发中需要经历的若干个测试级别，而且每一个级别都与一个开发级别相对应，但它忽略了测试的对象不应该仅仅只包括程序，或者说它没有明确地指出应该对软件的需求、设计进行测试，而这一点在 W 模型中得到了补充。W 模型强调了测试计划等工作的先行和对系统需求及系统设计的测试，但 W 模型和 V 模型一样，也没有专门对软件测试流程予以说明。因为事实上，随着软件质量要求越来越为大家所重视，软件测试也逐步发展成为一个独立于软件开发部的部门，就每一个软件测试的细节而言，都有一个独立的操作流程。例如，现在的第三方测试，就包含了从测试计划和测试用例编写，到测试实施以及测试报告编写的全过程，这个过程在 H 模型中得到了相应的体现，表现为测试是独立的。也就是说，只要测试前提具备了，就可以开始进行测试了。

因此，在实际的工作中，我们要灵活地运用各种模型的优点。在 W 模型的框架下，运用 H 模型的思想进行独立的测试，并同时将测试与开发紧密结合，寻找恰当的就绪点开始测试并反复迭代测试，最终保证按期完成预定目标。

6.1.2　软件测试过程中的活动及内容

软件测试贯穿整个软件开发周期，从需求提出到最终软件提交，软件测试过程中的关键活动包括提取测试需求、确定测试策略、制定测试计划、开展测试设计、执行测试用例、分析测试结果等。测试过程的主要内容是基于项目目标，制定测试计划，确定测试策略，选定测试方法，排定优先级，建立里程碑，组织测试资源(测试团队、软硬件环境等)等；然后，以测试计划为基础，明确测试需求、测试对象、测试目标以及功能与性能指标。最后，依据测试计划和测试设计，测试人员可以开展测试的相关活动，如开发和设计测试用例，选定测试工具，设计自动化测试框架，编写测试脚本，执行测试用例及测试脚本，分析测试结果，发现和报告缺陷。下面是具体的测试活动及内容。

1. 需求与规范管理(需求阶段)

需求分析阶段的工作就是对用户提出的需求进行分析并将每项分析结果作为测试设计阶段的输入。在该阶段，测试所要实现的目标就是确定测试需求。

(1) 由需求人员确定规范和需求，将规范和需求转发给开发经理、项目经理、相关开发人员和测试人员。

(2) 相关需求人员、项目经理对需求进行讨论，整理出重点，并将重点内容发给开发经理、项目经理、相关开发人员和测试人员。

(3) 对需求进行讨论，确定需求指标以及最终实现的需求和功能点。

(4) 项目经理根据需求和开会讨论结果编写需求说明，估算开发工作量。测试负责人或需求负责人对文档进行检查并修复完善。

(5) 测试负责人根据项目的需求说明来确定测试的需求，估算测试工作量。

2. 项目计划与测试计划(产品设计阶段)

根据开发估算的工作量进行项目计划，由项目经理给出计划表。由项目经理组织项目计划讨论会，讨论过程中，各开发负责人(包括测试负责人)对自己负责的模块或工作所需要的工作量进行评估，根据工作量和工程需求初步确定总体开发计划、测试计划和发布时间，项目经理根据工作量和需求制定项目计划。

(1) 由开发经理组织讨论会，各模块负责人评估工作量，根据工作量和需求初步确定开发计划、测试计划和发布时间。

(2) 项目经理根据估算工作量和需求编写项目计划。

(3) 测试负责人根据估算工作量和需求编写测试计划。

(4) 项目计划与测试计划完成后发送给开发经理、项目经理、相关开发人员和测试人员，认真阅读后将意见以邮件形式反馈给项目经理与测试负责人，进行二次修改，修改后召开小型会议进行讨论，最后定稿。

(5) 测试负责人确认所有相关文档已确认了评审。

3. 开发设计与评审(产品设计阶段)

该项工作以软件开发人员为主，测试人员可以参与其中以了解被测软件的设计情况。

4. 测试方案与评审(产品设计阶段)

(1) 项目设计阶段，测试负责人根据规范、功能列表和概要文档编写测试方案。

(2) 测试方案完成后，以邮件形式发送开发经理、项目经理、相关开发人员和测试人员。

(3) 对测试方案进行评审，最后将意见反馈给测试负责人，修改，最终确定测试方案。

5. 测试设计与评审(开发阶段)

(1) 进行详细的用例设计，包括功能测试、性能测试、压力测试等，以便发现更多的隐藏问题。

(2) 测试用例完成后，各负责人对用例进行评审。

6. 编码实现与单元测试(开发阶段、测试阶段)

开发设计编码，同时测试人员编写测试用例。

(1) 产品设计完成后，开发工程师进行编码。

(2) 编码完成后，测试工程师编写单元测试用例，进行单元测试。

(3) 项目经理根据实际情况对开发的编码组织代码复审，记录相关问题。

(4) 单元测试完成后，开发人员进行产品联调测试，并修改发现的问题。

7. 测试实施(测试阶段)

(1) 在测试环境中根据测试用例执行测试，在测试过程中发现问题，填写测试记录。
(2) 提交 bug。
(3) 开发人员修改 bug。
(4) 修改 bug 后，测试人员进行回归测试，直至关闭 bug。

8. 产品发布

测试产品达到测试计划所指定的产品质量目标和测试质量目标，进行最后一轮的确认测试，编写总体测试报告和性能测试报告，确认完成后进行产品发布。

6.1.3　软件测试过程度量

软件测试是保证软件质量的重要方法。随着软件开发规模的增大、复杂程度的增加，以寻找软件中的错误为目的的测试工作就更加复杂和困难。因此，为了尽可能多地找出程序中的错误，生产出高质量的软件产品，加强对测试工作的组织和管理就显得尤为重要，而其中很重要的一个方面是进行软件测试过程的度量。软件测试过程度量对于改进软件测试过程，提高软件测试效率具有重要意义。

1. 软件测试过程度量的意义

随着软件生产规模的日益增大，保证软件产品质量的软件测试工作越来越受到人们的重视，为更好地保证软件产品质量、更有效地执行软件测试工作，迫切需要对软件测试过程进行有效管理并逐步改善。目前，软件生产过程的管理模式也经历了完全依据管理者的经验到以数据为依据的量化管理这样一个逐步转换的过程。要对软件测试过程进行有效的管理，就需要有反映过程本质特性的过程数据，通过这些数据，测试过程的执行状况才能得到监控，测试过程改进的决策才能有的放矢。由于软件测试过程的不可见性，软件测试过程度量成为对软件测试过程进行量化管理不可或缺的一个环节。

软件测试过程度量是这样一种技术，它提取软件测试过程中可计量的属性，在测试过程进行中以一定频度不断地采集这些属性的值，并采用一些恰当的分析方法对得到的这些数据进行分析，建立可度量的指标，从而量化地评定测试过程的能力和性能，提高测试过程的可视性，帮助软件组织管理以及改进软件测试过程。

2. 软件测试过程度量指标

软件测试阶段的过程度量内容或条目比较多，包括软件测试进度、测试覆盖度、测试缺陷出现/到达曲线、测试缺陷累积曲线、测试效率等。在进行测试过程度量时，要基于软件规模度量(如功能点、对象点等)、复杂性度量、项目度量等方法，从测试广度、测试深度以及缺陷数三个不同的测度来完整度量测试过程。测试广度的测量提供了有多少需求(在所有需求的数目中)在某一时刻已经被测试，从而度量测试计划的执行、测试进度等状态；测试深度是对被测试覆盖的独立基本路径占程序中基本路径的总数百分比的测度，基本路径数目的度量可以用 McCabe 圈复杂度来计算；过程中收集的缺陷数，以及发现的、修正的和关闭的缺陷数在过程中的差异、发展趋势等，能为开发过程质量、开发资源额外投入、软件发布预测提供重要依据。

在 CMMI4 体系的测试过程中定义了 4 个度量指标：测试覆盖率、测试执行率、测试执行通过率、测试缺陷解决率。下面详细介绍这 4 个指标的含义及数据收集方法。

1) 测试覆盖率

测试覆盖率是指测试用例对需求的覆盖情况。计算公式：

测试覆盖率 = 已设计测试用例的需求数/需求总数

测试覆盖率从范围上说包括广度覆盖和深度覆盖；从内容上说包括用户场景覆盖、功能覆盖、功能组合覆盖、系统场景覆盖。

所谓广度，其含义是需求规格说明书中的每个需求项是否都在测试用例中得到设计。而所谓深度，通俗来说，是不使我们的测试设计流于表面，是否能够通过客户需求文档，挖掘出可能存在问题的地方。例如：重复单击某个按钮 10 次，或者依次执行新增、删除、新增同一数据的记录、再次删除该记录等操作。

在设计测试用例时，我们很少单独设计广度或深度方面的测试用例，而一般是结合在一起设计。为了从广度和深度上覆盖测试用例，我们需要考虑设计各种测试用例，如用户场景(识别最常用的 20%的操作)、功能点、功能组合、系统场景、性能、语句、分支等。在执行时，需要根据测试时间的充裕程度按照一定的顺序执行。通常是先执行用户场景的测试用例，然后再执行具体功能点、功能组合的测试用例。

对于测试覆盖率数据的收集，我们可以通过需求跟踪矩阵来实现。在需求跟踪矩阵中，测试人员填写系统测试用例列的数据。测试人员通过计算需求跟踪矩阵列出的需求数量，以及已设计测试用例的需求数量，可以快速地计算出测试覆盖率。通过需求跟踪矩阵，测试人员(包括项目组成员)可以很清楚地、快速地知道当前这个项目的测试覆盖情况。

2) 测试执行率

测试执行率，顾名思义，是指实际执行过程中确定已经执行的测试用例的比率。计算公式：

测试执行率 = 已执行的测试用例数/设计的总测试用例数

通常，我们设计的测试用例一般都是要执行的，即使是按模块来执行测试，该模块的测试执行率也一般是 100%，那么设置这个指标有何意义？

在实际测试过程中，经常有两种情况发生：①因为系统采用迭代方式开发，每次构建时都有不同的重点，包含不同的内容；②由于测试资源有限，不可能每次将所有设计的测试内容都全部测试完毕。由于这两种情况的存在，在每次执行测试时，我们会按照不同的测试重点和测试内容来安排测试活动，因此就有了“测试执行率”这个指标。

通常，我们的测试目标是确保 100%的测试用例都得到执行，即执行率为 100%。但是，正如前面所提到的，实际中可能存在非 100%的执行率。如果不能达到 100%的测试执行率，那么我们需要根据不同的情况制定不同的测试执行率标准——主要考虑风险、重要性、可接受的测试执行率。在考虑可接受的测试执行率时，就涉及测试用例执行顺序的问题。

在设计测试用例时，需要从广度和深度上尽可能地覆盖需求，所以我们就需要设计各种测试用例，如正常的测试用例、异常的测试用例、界面的测试用例等。但是在执行时，测试人员需要根据项目进度和测试时间的充裕程度，参考测试执行率标准，将测试用例按照一定的顺序执行。通常是先执行用户场景对应的测试用例，再执行具体功能点、功能组

合的测试用例，完成这些测试后，再进行其他测试，如系统场景、性能、语句等测试。

例如，某项目共设计了 280 个测试用例。该项目某一阶段的测试共分四个版本，其中有一个版本执行了 134 个测试用例，那么该版本的测试执行率为 47.9%。

3) 测试执行通过率

测试执行通过率，是指在实际执行的测试用例中，执行结果为“通过”测试用例的比率。 计算公式：

测试执行通过率 = 执行结果为“通过”的测试用例数/实际执行的测试用例总数

我们可以针对所有计划执行的测试用例进行衡量，可以细化到具体模块，用于对比各个模块的测试用例执行情况。

为了得到测试执行通过率数据，我们在测试执行时，需要在测试用例副本中记录下每个测试用例的执行结果，然后在当前版本执行完毕，或者定期(如每周)统计当前测试执行数据。通过原始数据的记录与统计，我们可以快速地得到当前版本或当前阶段的测试执行通过率。

4) 缺陷解决率

缺陷解决率，是指某个阶段已关闭缺陷占缺陷总数的比例。缺陷关闭操作包括以下两种情况：①正常关闭。缺陷已修复，且经过测试人员验证通过；②强制关闭。诸如重复的缺陷、由于外部原因造成的缺陷、暂时不处理的缺陷、无效的缺陷等，这类缺陷经过确认后，可以强制关闭。计算公式：

缺陷解决率 ＝ 已关闭的缺陷/缺陷总数

在项目过程中，在开始时缺陷解决率上升很缓慢，随着测试工作的开展，缺陷解决率逐步上升，在版本发布前，缺陷解决率将趋于 100%。一般来说，在每个版本对外发布时，缺陷解决率都应该达到 100%。也就是说，除了已修复的缺陷需要进行验证外，其他需要强制关闭的缺陷必须经过确认，且有对应的应对措施。可以将缺陷解决率作为测试结束和版本发布的一个标准。如果有部分缺陷仍处于打开或已处理状态，那么原则上来说，该版本是不允许发布的。

可以通过缺陷跟踪工具定期(如每周)收集当前系统的缺陷数、已关闭缺陷数，通过这两项数据，即可绘制出整个项目过程或某个阶段的缺陷解决率曲线。

实际上，测试度量指标不仅仅只包括上述四个，在实际工作中，还会用到验证不通过率、缺陷密度等指标。收集这些数据的目的是对测试过程进行量化管理。但是，简单收集度量数据不是目的，通过对数据的分析、预防问题、对问题采取纠正措施，减少风险才是目的。

3. 软件测试过程度量原则

对软件测试过程质量度量应该遵循四项原则：①要制定明确的度量目标；②建立软件测试过程质量度量的指标体系，度量指标的定义应该具有一致性、客观性；③度量的方法应该尽可能简单、可计算；④度量数据的收集应该尽可能自动化。

6.1.4　软件测试过程成熟度

从本质上来说，无论是传统的软件测试，还是面向整个开发过程的基于生命周期的软件测试，所针对的测试对象都是软件产品、半成品或过程工作产品，所报告的测试结果也只是为了识别出现阶段产品的缺陷，并加以纠正以支持下一阶段的开发工作。

从软件开发组织的长远发展来看，仅仅做到这些还是不够的。软件测试作为软件质量保证的一种重要手段，不仅要能够识别软件产品的缺陷并加以改正，还应该在软件测试中结合统计技术方法， 给出对软件开发过程的度量，从而支持组织对软件开发过程的评估和改进。由美国国防部和卡耐基-梅隆大学的软件工程研究所联合开发的软件能力成熟度模型CMM，正是从软件过程改进和评估的角度出发，对软件开发中的测试技术给出充分的支持和扩充。但仅仅如此是不够的，因为在 CMM 中，软件测试并没有予以充分定义，软件测试成熟度概念没有提及，软件测试过程的改进没有充分说明，在关键过程域 KPA 中没有定义测试问题，与质量相关的测试问题(如可测性、充分测试标准、测试计划等方面)也没有满意的阐述。仅在第三级的软件产品工程(Software Product Engineering，SPE)KPA 中提及软件测试职能，但对于如何有效提高机构的测试能力和水平没有提供相应指导，这显然是有问题的。

为此，许多研究机构和测试服务机构从不同角度出发，提出有关软件测试方面的能力成熟度模型，作为 SEI-CMM 的有效补充，比较有代表性的包括：美国国防部提出一条CMM软件评估和测试 KPA 建议；Gelper 博士提出测试支持模型(Testing Support Model，TSM)评估测试小组所处环境对于他们的支持程度；Burgess/Drabick I.T.I.公司提出的测试能力成熟度模型(Testing Capability Maturity Model)则提供了与 CMM 完全一样的 5 级模型；Burnstein 博士提出了测试成熟度模型(TMM)，依据 CMM 的框架提出测试的 5 个不同级别，关注于测试的成熟度模型。它描述的测试过程，是项目测试部分得到良好计划和控制的基础。TMM 测试成熟度分为 5 级别，5 个成熟度级别递增。

1. 初始级

TMM初始级软件测试过程的特点是测试过程无序，有时甚至是混乱的，几乎没有妥善的定义。初始级中软件的测试与调试常常被混为一谈，软件开发过程中缺乏测试资源、工具以及训练有素的测试人员。初始级的软件测试过程没有定义成熟度目标。

2. 定义级

在 TMM 的定义级中，测试已具备基本的测试技术和方法，软件的测试与调试已经明确地被区分开。这时，测试被定义为软件生命周期中的一个阶段，它紧随在编码阶段之后。但在定义级中，测试计划往往在编码之后才得以制定，这显然有悖于软件工程的要求。

在 TMM 的定义级中需要实现 3 个成熟度目标：制定测试与调试目标，启动测试计划过程，制度化基本的测试技术和方法。

1) 制定测试与调试目标

软件组织必须清晰地区分软件开发的测试过程与调试过程，识别各自的目标、任务和活动。正确区分这两个过程是提高软件组织测试能力的基础。与调试工作不同，测试工作是一种有计划的活动，可以进行管理和控制。这种管理和控制活动需要制定相应的策略和

政策，以确定和协调这两个过程。

制定测试与调试目标包含 5 个子成熟度目标：①分别形成测试组织和调试组织，并有经费支持；②规划并记录测试目标；③规划并记录调试目标；④将测试和调试目标形成文档，并分发至项目；⑤将测试目标反映到测试计划中。

2) 启动测试计划过程

制定计划是使一个过程可重复、可定义和可管理的基础。测试计划应包括测试目的、风险分析、测试策略以及测试设计规格说明和测试用例。此外，测试计划还应说明如何分配测试资源，如何划分单元测试、集成测试、系统测试和验收测试的任务。启动测试计划过程包含 5 个子目标：①建立组织内的测试计划组织并予以经费支持；②建立组织内的测试计划政策框架并予以管理上的支持；③开发测试计划模板并分发至项目的管理者和开发者；④建立一种机制，使用户需求成为测试计划的依据之一；⑤评价、推荐和获得基本的计划工具并从管理上支持工具的使用。

3) 制度化基本的测试技术和方法

为改进测试过程能力，组织需要应用基本的测试技术和方法，并说明何时和怎样使用这些技术、方法和支持工具。将基本测试技术和方法制度化有两个子目标：①在组织范围内成立测试技术组，研究、评价和推荐基本的测试技术和测试方法，推荐支持这些技术与方法的基本工具；②制定管理方针以保证在整个组织范围内一致使用所推荐的技术和方法。

3. 集成级

在集成级，测试不仅仅是跟随在编码阶段之后的一个阶段，它已被扩展成与软件生命周期融为一体的一组已定义的活动。测试活动遵循软件生命周期的 V 模型。测试人员在需求分析阶段便开始着手制定测试计划，并根据用户或客户需求建立测试目标，同时设计测试用例并制定测试通过准则。在集成级，应成立软件测试组织，提供测试技术，关键的测试活动应有相应的测试工具予以支持。在该测试成熟度等级，没有正式的评审程序，没有建立质量过程和产品属性的测试度量。集成级要实现 4 个成熟度目标，它们分别是：建立软件测试组织，制定计划，软件全生命周期测试，控制和监视测试过程。

1) 建立软件测试组织

软件测试的过程及质量对软件产品的质量有直接影响。由于测试往往是在时间紧、压力大的情况下所完成的一系列复杂的活动，因此应由训练有素的专业人员组成测试组。测试组要完成与测试有关的多种活动，包括负责制定测试计划，实施测试执行，记录测试结果，制定与测试有关的标准和测试度量，建立测试数据库，测试重用，测试跟踪以及测试评价等。建立软件测试组织要实现 4 个子目标：①建立整个组织范围内的测试组，并得到上级管理层的领导和各方面的支持，包括经费支持；②定义测试组的作用和职责；③由训练有素的人员组成测试组；④建立与用户或客户的联系，收集他们对测试的需求和建议。

2) 制定计划

为高效率地完成测试工作，测试人员必须经过适当的培训。制定技术培训规划有 3 个子目标：①制定组织的培训计划，并在管理上提供包括经费在内的支持；②制定培训目标

和具体的培训计划；③成立培训组，配备相应的工具、设备和教材。

3) 软件全生命周期测试

提高测试成熟度和改善软件产品的质量都要求将测试工作与软件生命周期中的各个阶段联系起来。该目标有 4 个子目标：①将测试阶段划分为子阶段，并与软件生命周期的各阶段相联系；②基于已定义的测试子阶段，采用软件生命周期的 V 模型；③制定与测试相关的工作产品的标准；④建立测试人员与开发人员共同工作的机制，这种机制有利于促进将测试活动集成于软件生命周期中。

4) 控制和监视测试过程

为控制和监视测试过程，软件组织需要采取相应措施，如制定测试产品的标准、制定与测试相关的偶发事件的处理预案、确定测试里程碑、确定评估测试效率的度量、建立测试日志等。控制和监视测试过程有 3 个子目标：①制定控制和监视测试过程的机制和政策；②定义、记录并分配一组与测试过程相关的基本测量；③开发、记录并文档化一组纠偏措施和偶发事件处理预案，以备实际测试严重偏离计划时使用。

在 TMM 的定义级，在测试过程中引入计划能力；在 TMM 的集成级，在测试过程中引入控制和监视活动。两者均为测试过程提供了可见性，为测试过程持续进行提供保证。

4. 管理和测量级

在管理和测量级，测试活动除测试被测程序外，还包括软件生命周期中各个阶段的评审、审查和追查，使测试活动涵盖软件验证和软件确认活动。根据管理和测量级的要求，软件工作产品以及与测试相关的工作产品，如测试计划、测试设计和测试步骤都要经过评审。因为测试是一个可以量化并度量的过程，为了测量测试过程，测试人员应建立测试数据库。收集和记录各软件工程项目中使用的测试用例，记录缺陷并按缺陷的严重程度划分等级。此外，所建立的测试规程应能够支持软件组织对测试过程的控制和测量。管理和测量级有 3 个要实现的成熟度目标：建立组织范围内的评审程序，建立测试过程的测量程序和软件质量评价。

1) 建立组织范围内的评审程序

软件组织应在软件生命周期的各阶段实施评审，以便尽早有效地识别、分类和消除软件中的缺陷。建立评审程序有 4 个子目标：①管理层要制定评审政策来支持评审过程；②测试组和软件质量保证组要确定并文档化整个软件生命周期中的评审目标、评审计划、评审步骤以及评审记录机制；③评审项由上层组织指定；④培训参加评审的人员，使他们理解和遵循相关的评审政策、评审步骤。

2) 建立测试过程的测量程序

测试过程的测量程序是评价测试过程质量、改进测试过程的基础，对监视和控制测试过程至关重要。测量包括测试进展、测试费用、软件错误和缺陷数据以及产品测量等。建立测试过程的测量程序有 3 个子目标：①定义组织范围内的测试过程的测量政策和目标；②制定测试过程的测量计划，测量计划中应给出收集、分析和应用测量数据的方法；③应用测量结果，制定测试过程的改进计划。

3) 软件质量评价

软件质量评价内容包括定义可测量的软件质量属性，定义评价软件工作产品的质量目标等项工作。软件质量评价有两个子目标：①管理层、测试组和软件质量保证组要制定与质量有关的政策、质量目标和软件产品质量属性；②测试过程应是结构化、已测量和已评价的，以保证达到质量目标。

5. 优化、预防缺陷和质量控制级

由于本级别的测试过程是可重复、已定义、已管理和已测量的，因此软件组织能够优化调整和持续改进测试过程。测试过程的管理为持续改进产品质量和过程质量提供指导，并提供必要的基础设施。优化、预防缺陷和质量控制级有 3 个要实现的成熟度目标：应用过程数据预防缺陷，质量控制，优化测试过程。

1) 应用过程数据预防缺陷

这时的软件组织能够记录软件缺陷，分析缺陷模式，识别错误根源，制定防止缺陷再次发生的计划，提供跟踪这种活动的办法，并将这些活动贯穿于整个组织的各个项目中。应用过程数据预防缺陷有 4 个成熟度子目标：①成立缺陷预防组；②识别和记录在软件生命周期各阶段引入的软件缺陷和消除的缺陷；③建立缺陷原因分析机制，确定缺陷原因；④管理人员、开发人员和测试人员互相配合，制定缺陷预防计划，防止已识别的缺陷再次发生。缺陷预防计划要具有可跟踪性。

2) 质量控制

在本级别，软件组织通过采用统计采样技术，测量组织的自信度，测量用户对组织的信赖度以及设定软件可靠性目标来推进测试过程。为了加强软件质量控制，测试组和质量保证组要有负责质量的人员参加，他们应掌握能减少软件缺陷和改进软件质量的技术和工具。支持统计质量控制的子目标有：①软件测试组和软件质量保证组建立软件产品的质量目标，如产品的缺陷密度、组织的自信度以及可信赖度等；②测试管理者要将这些质量目标纳入测试计划；③培训测试组学习和使用统计学方法；④收集用户需求以建立使用模型。

3) 优化测试过程

在测试成熟度的最高级，已能够量化测试过程。这样就可以依据量化结果来调整测试过程，不断提高测试过程能力，并且软件组织具有支持这种能力持续增长的基础设施。基础设施包括政策、标准、培训、设备、工具以及组织结构等。优化测试过程包含：①识别需要改进的测试活动；②实施改进；③跟踪改进进程；④不断评估所采用的与测试相关的新工具和新方法；⑤支持技术更新。

测试过程优化所需子成熟度目标包括：①建立测试过程改进组，监视测试过程并识别需要改进的部分；②建立适当的机制以评估改进测试过程能力和测试成熟度的新工具和新技术；③持续评估测试过程的有效性，确定测试终止准则。终止测试的准则要与质量目标相联系。

6.1.5　软件测试过程改进

软件测试过程也就是软件测试生命周期，它严重影响着软件开发的效率和软件产品的

质量。测试技术解决了测试采用的方法和技术问题，测试管理保证了各项测试活动的顺利开展。软件测试过程改进主要着眼于合理调整各项测试活动的时序关系，优化各项测试活动的资源配置以及实现各项测试活动效果的最优化。在软件测试过程中，过程改进被赋予举足轻重的地位，在测试计划、实施、检查、改进的循环中，过程改进既是一次质量活动的终点，又是下次质量活动的原点，起着承上启下的作用，因此软件测试过程改进对于软件质量的提高相当重要。

1. 软件测试过程改进的概念

测试过程的改进对象应该包括三个方面：组织、技术和人员。需要对组织给予特别关注，因为过程都是基于特定的组织架构建设的，而且组织设置是否合理对过程的好坏有决定性的影响。

1) 软件测试过组织架构问题

软件测试组织的不良架构通常表现在：①没有恰当的角色追踪项目进展；②没有恰当的角色进行缺陷控制、变更和版本追踪；③项目在测试阶段效率低下、过程混乱；④只有测试经理了解项目，项目成了个人的项目，而不是组织的项目；⑤关心进度，而忘了项目的另外两个要素——质量和成本。

上述问题可从组织上找出原因。因此在测试过程改进中可以先将测试从开发活动中分离出来，把缺陷控制、版本管理和变更管理从项目管理中分离出来。此外，需要给测试经理赋予明确的职责和目标。技术的改进包括对流程、方法和工具的改进，包括组织或项目对流程进行明确的定义，杜绝随机过程，引入统一的管理方法，并使用标准的经过组织认可的工具和模板。人员的改进主要是指对企业文化的改进，将促使建立高效率的团队和组织。

2) 软件测试过程改进战略

由于测试过程改进是一项长期的、没有终点的活动，而且要获得改进过程的收益也是长期的过程，因此在起步实施测试过程改进时，要充分考虑战略，并根据公司的战略目标确定测试部门的战略，描绘远景。将测试过程改进与公司战略目标相联系，是改进成功实施的必要条件，也是各公司在实施测试过程改进中获得的最佳实践。在研究过程中，组织的规划通常包括如下内容：

(1) 绘制远景：如提升管理成熟度，提高测试生产率，促使部门测试能力达到公司领先水平。

(2) 战略分析：如在部门内制定三年计划。以内部人员为主，引入适当的培训，通过一年半到两年的内部过程，使 verification/validation(验证/确认)及其他相关过程得到改进并达到 CMMI3 成熟度，适时进行评估，最终目标为 CMMI4。

(3) 优缺点评估：上述战略方法的优点在于前期以内部改进为宗旨，避免了拔苗助长带来的风险，可以使过程改进更符合组织的实际情况。缺点是不以正式评估作为目标，可能导致领导关注力度减弱、过程改进的动力不足，因此需要过程改进的负责人具有坚韧的斗志和持之以恒的信念。

3) 软件测试过程改进策略选择举例

在改进的不同时期和阶段，选择的策略也不同，组织应根据实际情况进行选择，下面列举在研究过程中收集的可供参照的主要策略方法。

(1) 重诊断，轻评估。要以诊断和解决测试过程中的实际问题作为测试过程改进的目的，不能盲目追求商业评估。在以往实施 ISO 9000 的过程中曾发现，组织拿证书的愿望常常会冲淡过程改进的真正目的。

(2) 实施制度化的同时，建设企业文化。实施全面制度化的管理是过程改进的有效保障，制度和组织文化总是互相依存，没有良好的文化保障，制度化将困难重重；而没有制度的支撑，文化也将是无根之木。

(3) 引入软件工具。推行配置、自动化测试和缺陷跟踪等工具，将有效地分解事务性工作，可以缓解人力资源不足的困难。常见的过程管理方面的工具包括 IBM Rational 公司的 CleaerCase、ClearQuest，CA 公司的 CCC/Harvest 以及 HP 公司的 ALM 及 UFT 等。

(4) 建设管理和工程基础。为了解决基础薄弱的问题，需要在测试过程改进前期对相关部门和员工进行基础管理和基本软件工程的课程培训。

(5) 发动全员参与。全员参与可以分三个层面来理解：第一，站在高于项目管理的层面；第二，站在项目管理的层面；第三，站在开发人员和测试人员的层面。充分调动各方人员的积极性。

(6) 现有过程的复用。该原则可以充分利用现有过程的合理部分，提高被改进过程的可接受程度和使用价值。

2. 软件测试过程改进的具体方法

过程改进在软件测试过程中占有举足轻重的位置，因此为了更好地保证软件质量，测试过程改进是测试人员经常要做的事情，下面列出了一些软件测试过程改进的具体方法。

1) 调整测试活动的时序关系

在软件测试过程的测试计划中，不恰当的测试时序会引起误工和测试进度失控。例如，具体到某个工程实践中，有些测试活动是可以并行的，有些测试活动是可以归并完成的，有些测试活动在时间上存在线性关系，等等。所有这些一定要区分清楚并且要做最优化调整，否则会对测试进度产生不必要的影响。

2) 优化测试活动资源配置

在软件测试过程中，必然会涉及人力、设备、场地、软件环境与经费等资源，那么如何合理地调配各项资源给相关的测试活动是非常值得斟酌的，否则会引起误工和测试进度的失控。在测试资源配置中最常见的人力资源的调配，测试部门如能深入了解员工的专长与兴趣所在，在进行人员分配时，根据各自的特点进行分配，就能对测试活动的开展起到事半功倍的效果。

3) 提高测试计划的指导性

测试计划的指导性是指测试计划的执行能力。在软件测试过程中，很多时候实际的测试和测试计划是脱节的，或者说很大程度上没有按照测试计划去执行。测试计划的完成不仅仅是起草测试大纲，而是为了确保测试大纲以及计划的内容能真正被执行、真正用于指

导测试工作，为了更好地完成测试活动，保证软件的质量。

4) 确立合理的度量模型和标准

在测试过程改进中，测试过程改进小组应根据企业与项目的实际情况制定适合自己公司的质量度量模型和标准，做出符合自己公司发展策略的投入。但是质量度量模型和标准的确立不是马上就可以进行的，而是测试过程改进小组随着测试过程的进行不断实践、不断总结、不断改进的。质量度量模型和标准一旦确立，很多测试活动就不至于陷入过度测试或测试不够的尴尬状态中，使得测试活动在公司与项目不断发展变化的氛围中保持动态平衡。

5) 提高覆盖率

覆盖率越高，表明测试的质量越高。覆盖率包括内容的覆盖和技术的覆盖。内容的覆盖指的是起草测试计划、设计测试用例、执行测试用例和跟踪软件缺陷，内容覆盖率越高，就越能避免故障被遗漏的情况；技术的覆盖指一项技术指标要尽可能做到测试技术的覆盖，采用科学的方法来验证某项指标，可以更好地保证产品的质量。

除了上面讲的测试过程改进的具体方法外，我们还应注意如下事项：一是必须注意过程改进是跟公司的发展战略相关的，否则只会对测试过程产生不利的影响；二是测试过程的改进并不意味着必须投入大笔资金；三是在测试过程改进中可以参照 CMM 模型与技术。

6.2　软件测试过程管理

现代软件测试过程管理不仅锁定在测试阶段，软件测试过程管理在各个阶段的具体内容虽然是不同的，但在每个阶段，测试任务的最终完成都要经过从计划、设计、执行到结果分析、总结等一系列相同的步骤，这构成软件测试的一个基本过程。通过软件测试过程管理，我们要尽量达到测试成本最小化、测试流程和测试内容完备化、测试手段可行化和测试结果实用化的理想目标。

软件测试是软件工程的一个子过程，为使软件测试工作系统化、工程化，必须合理组合测试过程管理，包括签订第三方独立测试合同、制定测试计划、组织项目人员、建立项目环境、监控项目进展等。软件测试过程主要包括软件测试项目的启动、测试需求分析、测试计划、测试用例设计、测试执行、测试结果的审查和分析等活动及内容，如图6-4所示。

1. 测试项目启动

首先要确定项目组长，只要把项目组长确定下来，就可以组建整个测试小组，并可以和开发等部门开展工作。接着参加有关项目计划、分析和设计的会议，获得必要的需求分析、系统设计文档，以及相关产品/技术知识的培训和转移。

2. 测试需求分析

测试需求通常是以软件开发需求为基础进行分析，通过对开发需求的细化和分解，形成可测试的内容。因此，测试需求分析包括五个部分：①明确需求的范围；②明确每一个功能的业务处理过程；③不同的功能点与业务的组合；④挖掘显式需求背后的隐式需求；

⑤测试需求应全部覆盖已定义的业务流程，以及功能和非功能方面的需求。

图 6-4　软件测试过程图

3. 制定测试计划

确定测试范围、测试策略和方法，以及对风险、日程表、资源等进行分析和估计。

4. 测试设计和测试开发

制定测试的技术方案、设计测试用例、选择测试工具、写测试脚本等。测试用例设计要事先做好各项准备，才开始进行，最后还要让其他部门审查测试用例。

5. 测试实施和执行

建立或设置相关的测试环境，准备测试数据，执行测试用例，对发现的软件缺陷进行报告、分析、跟踪等，测试执行没有很高的技术性，但却是测试的基础，直接关系到测试的可靠性、客观性和准确性。

6. 测试结果的审查和分析

当测试执行结束后，对测试结果要进行整体或综合的分析，以确定软件产品质量的当前状态，为产品的改进或发布提供数据和依据。从管理来讲，要做好测试结果的审查和分析会议，以及做好测试报告或质量报告的写作、审查。

根据测试需求、测试计划，对测试过程中的每个状态进行记录、跟踪和管理，并提供相关的分析和统计功能，生成和打印各种分析统计报表。通过对详细记录进行分析，形成较为完整的软件测试管理文档，避免同样的错误在软件开发过程中再次发生，从而提高软件开发质量。软件测试过程的管理如图 6-5 所示。

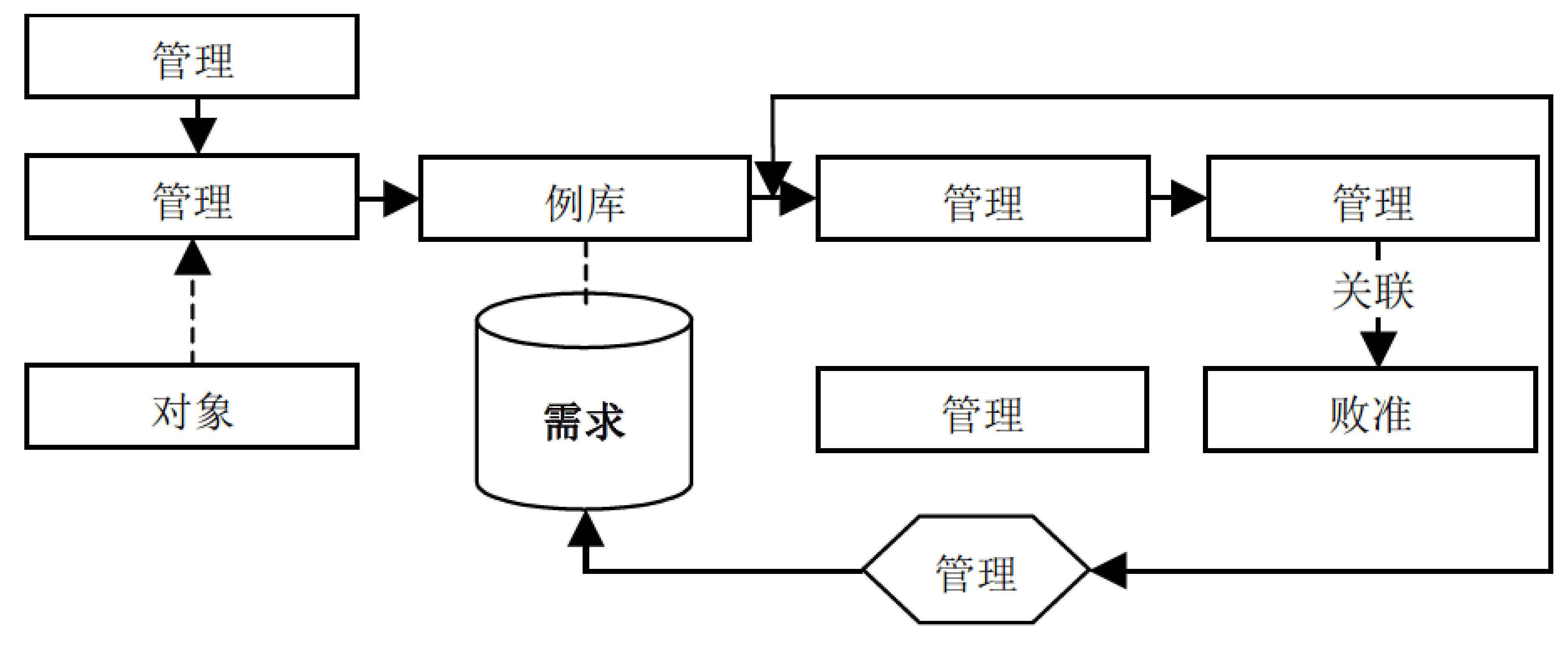


图 6-5　软件测试过程管理图

6.2.1　软件测试过程管理的理念

软件测试生命周期模型或软件测试过程模型为我们提供了软件测试的流程和方法，为测试过程管理提供了依据。由于测试过程管理牵涉的范围非常广泛，包括过程定义、人力资源管理、风险管理等，因此我们仅从前面介绍的软件测试过程模型(如 W 模型)来介绍软件测试过程管理的思想。

1. 尽早测试

“尽早测试”是从 W 模型中抽象出来的理念。我们说的测试并不是在代码编写完成之后才开展的工作，测试与开发是两个相互依存的并行过程，测试活动在开发活动的前期已经开展。

“尽早测试”包含两方面的含义：第一，测试人员早期参与软件项目，及时开展测试的准备工作，包括编写测试计划、制定测试方案以及准备测试用例；第二，尽早地开展测试执行工作，一旦代码模块完成就应该及时开展单元测试，一旦代码模块被集成，成为相对独立的子系统，便可以开展集成测试，一旦有 build 提交，便可以开展系统测试工作。

由于及早地开展了测试准备工作，测试人员能够于早期了解测试的难度、预测测试的风险，从而有效提高了测试效率，规避测试风险；由于及早地开展测试执行工作，测试人员能尽早地发现软件缺陷，大大降低了 bug 修复成本。但是需要注意，“尽早测试”并非盲目提前测试活动，测试活动开展的前提是达到必需的测试就绪点。

2. 全面测试

软件是程序、数据和文档的集合，那么对软件进行测试，就不仅仅是对程序的测试，还应包括对软件“副产品”的全面测试，这是 W 模型中一个重要的思想。需求文档、设计文档作为软件的阶段性产品，直接影响到软件的质量。阶段产品质量是软件质量的量的积累，不能把握这些阶段产品的质量将导致最终软件质量的不可控。

“全面测试”包含两层含义：第一，对软件的所有产品进行全面的测试，包括需求、设计文档、代码、用户文档等；第二，软件开发及测试人员(有时包括用户)全面地参与到测试工作中，例如对需求的验证和确认活动，就需要开发人员、测试人员及用户的全面参与，毕竟测试活动并不仅仅是保证软件运行正确，同时还要保证软件满足用户的需求。

“全面测试”有助于全方位把握软件质量，尽最大可能地排除造成软件质量问题的因

素，从而保证软件满足质量需求。

3. 全过程测试

在 W 模型中充分体现的另一个理念就是“全过程测试”。双 V 过程图形象地表明了软件开发与软件测试的紧密结合，这就说明软件开发和测试过程会彼此影响，这就要求测试人员对开发和测试的全过程进行充分的关注。

“全过程测试”包含两层含义：第一，测试人员要充分关注开发过程，对开发过程的各种变化及时做出响应，例如开发进度的调整可能会引起测试进度及测试策略的调整，需求的变更会影响到测试的执行，等等；第二，测试人员要对测试的全过程进行全程跟踪，例如建立完善的度量与分析机制，通过对自身过程的度量，及时了解过程信息、调整测试策略。

“全过程测试”有助于及时应对项目变化，降低测试风险，同时对测试过程的度量与分析也有助于把握测试过程，调整测试策略，便于测试过程的改进。

4. 独立的、迭代的测试

我们知道，软件开发中的瀑布模型只是一种理想状况。为适应不同的需要，人们在软件开发过程中摸索出了螺旋、迭代等诸多模型。在这些模型中，需求、设计、编码工作可能是重叠并反复进行的，这时的测试工作也将是迭代和反复的。如果不能将测试从开发中抽象出来进行管理，势必使测试管理陷入困境。

软件测试与软件开发是紧密结合的，但并不代表测试是依附于开发的一个过程，测试活动是独立的。这正是 H 模型所主导的思想。“独立的、迭代的测试”着重强调测试的就绪点，也就是说，只要测试条件成熟，测试准备活动完成，测试的执行活动就可以开展。

所以，我们在遵循尽早测试、全面测试、全过程测试理念的同时，应当将测试过程从开发过程中适当地抽象出来，作为一个独立的过程进行管理。时刻把握“独立的、迭代的测试”的理念，减小因开发模型的繁杂而给测试管理工作带来的不便。对于软件过程中不同阶段的产品和不同的测试类型，只要测试准备工作就绪，就可以及时开展测试工作，把握产品质量。

6.2.2　软件测试计划与测试需求

软件测试计划是指导测试过程的纲领性文件，包含产品概述、测试策略、测试方法、测试区域、测试配置、测试周期、测试资源、测试交流、风险分析等内容。借助软件测试计划，参与测试的项目成员，尤其是测试管理人员，可以明确测试任务和测试方法，保持测试实施过程的顺畅沟通，跟踪和控制测试进度，应对测试过程中的各种变更。测试计划在需求活动一开始就要着手编写，随着开发过程的逐步展开添加内容，在编程活动和单元测试活动之后完成测试计划的编写。测试计划按国家标准或行业标准规定的格式和内容编写。

另外，测试计划最关键的一步就是进行测试需求分析，测试需求是测试工作的基础。

1. 软件测试计划的制定

测试计划要针对测试目的规定测试的任务、所需的各种资源和资金、人员及时间投入、

人员角色的安排以及预见可能出现的问题和风险等，以指导测试的执行，最终实现测试的目标，保证软件产品的质量。

1) 制定测试计划的目的

编写测试计划的目的是：

① 为测试各项活动制定一个现实可行的、综合的计划，包括每项测试活动的对象、范围、方法、进度和预期结果；

② 为项目实施建立一个组织模型，并定义测试项目中每个角色的责任和工作内容；

③ 开发有效的测试模型，能正确地验证正在开发的软件系统；

④ 确定测试所需要的时间和资源，以保证其可获得性、有效性；

⑤ 确立每个测试阶段测试完成以及测试成功的标准、要实现的目标；

⑥ 识别出测试活动中的各种风险，并消除可能存在的风险，降低那些不可能消除的风险所带来的损失。

测试计划是一个重要文档，因此在形成测试计划的过程中要对测试计划和测试设计进行检查，当发现错误和遗漏时能在开发过程的早期对测试计划进行必要的增加和修改，减少测试设计的错误。因此，形成一份完整的、精确的和全面的测试计划需要经过计划、准备、检查、修改和继续五个步骤。

2) 测试计划阶段的划分

测试计划不可能一气呵成，而是要经过计划初期、起草、讨论、审查等不同阶段，才能将测试计划制定好。而且，不同的测试阶段(集成测试、系统测试、验收测试等)或测试任务(安全性测试、性能测试、可靠性测试等)都可能要有具体的测试计划。

(1) 计划初期是收集整体项目计划、需求分析、功能设计、系统原型、用例报告等文档或信息，理解用户的真正需求，了解技术难点、弱点或新的技术，和其他项目相关人员交流，在各个主要方面达到一致的理解。

(2) 测试计划最关键的一步就是确定测试需求和测试层次。将软件分解成单元，对各个单元写成测试需求，测试需求也是测试设计和开发测试用例的基础，并用来衡量测试覆盖率的重要指标。

(3) 确定软件测试目标，并使其可以量化、度量和相对集中。可通过对用户需求文档和设计规格文档的分析，来确定被测软件的质量要求和测试需要达到的目标。

(4) 计划起草。根据计划初期所掌握的各种信息、知识，确定测试策略，设计测试方法，完成测试计划的框架。

(5) 内部审查。在提供给其他部门讨论之前，先在测试小组/部门内部进行审查。

(6) 讨论和修改计划。召开有需求分析、设计、开发人员参加的计划讨论会议，测试组长对测试计划的设计思想、策略做较详细的介绍，并听取大家对测试计划中各个部分的意见，进行讨论交流。

(7) 测试计划的多方审查。项目中的每个人都应当参与审查(即市场、开发、支持、技术协作及测试人员)。计划的审查是必不可少的，尽管测试工程师努力地完成对产品的全面定义，但出自测试工程师的定义不一定是完整或准确的。此外，就像开发者很难测试自己的代码那样，测试工程师也很难评估自己的测试计划。每一个计划审查者都可能根据其经

验及专长提出修改建议，有时还能提供测试工程师在组织产品定义时不具备的信息。

(8) 测试计划的定稿和批准。在计划讨论、审查的基础上，综合各方面的意见，就可以完成测试计划书，然后报给测试经理或 QA 经理，得到批准，方可执行。测试计划不仅是软件产品当前版本而且还是下一个版本的测试设计的主要信息来源，在进行新版本测试时，可以对原有的软件测试计划书做修改，但要经过严格审查。

3) 测试计划的要点

软件测试计划的内容主要包括产品基本情况、测试需求说明、测试策略和记录、测试资源配置、计划表、问题跟踪报告、测试计划的评审、结果等。除了产品基本情况、测试需求说明、测试策略等，测试计划的焦点集中在以下几个方面：

(1) 计划的目的：项目的范围和目标，各阶段的测试范围、技术约束和管理特点。

(2) 项目估算：使用的历史数据，使用的评估技术，工作量、成本、时间估算依据。

(3) 风险计划：测试可能存在的风险分析、识别，以及风险的回避、监控、管理。

(4) 日程：项目工作分解结构，并采用时限图、甘特图等方法制定时间/资源表。

(5) 项目资源：人员、硬件和软件等资源的组织和分配，人力资源是重点，而且日程安排紧密联系。

(6) 跟踪和控制机制：质量保证和控制、变化管理和控制等。测试计划书的内容也可以按集成测试、系统测试、验收测试等阶段去组织，为每一个阶段制定一份计划书，也可以为每个测试任务/目的(安全性测试、性能测试、可靠性测试等)制定特别的计划书。

此外，可对上述测试计划书的每项内容，制定具体的实施计划，如对每个阶段的测试重点、范围、所采用的方法、测试用例设计的思想、提交的内容等进行细化，供测试项目组的内部成员使用。对于一些重要的项目，要形成一系列的计划书，如测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

4) 测试计划的编写内容

我们要按照国家标准或有关行业标准编写测试计划，测试计划要提供被测软件的背景信息、测试目标、测试步骤、测试数据整理以及评估准则。具体包括：①测试环境(在不同的条件下进行测试，所得到的结果是不同的，在测试计划中，测试环境是指用户使用环境或业务运行环境)；②测试基本原理和策略；③测试计划阶段划分；④测试计划要点；⑤功能描述和功能覆盖说明；⑥测试用例清单，说明每个测试用例测什么；⑦测试开始准则和退出准则。

每个测试用例的序言至少包括下面这些信息：测试用例说明和用途，设置需求(输入/输出)，运行测试用例的操作命令，正常和异常信息，编写测试用例的作者。

2. 软件测试需求分析

一个成功的测试项目，首先必须了解测试规模、复杂程度与可能存在的风险，这些都需要通过详细的测试需求来了解。测试需求不明确，只会造成获取的信息不正确，无法对被测软件有一个清晰全面的认识，测试计划就毫无根据可言；另外，测试需求越详细精准，对被测软件的了解越深，对所要进行的任务内容就越清晰，就更有把握保证测试的质量与

进度；最后，软件测试需求是开发测试用例的依据，是衡量测试覆盖率的重要指标。

在进行需求分析时我们要把握 3 条原则：①制定的测试需求项必须是可核实的，即它们必须有可观察、可评测的结果，无法核实的需求不是测试需求；②测试需求应指明满足需求的正常的前置条件，同时也要指明不满足需求时的出错条件；③测试需求不涉及具体的测试数据，测试数据的设计是测试设计环节应解决的内容。

测试需求分析过程实际上就是测试需求的收集、分析与评审过程，如图 6-6 所示。

1) 软件测试需求的收集

测试需求通常是以被测对象的软件需求为原型进行分析而转变过来的，但测试需求并不等同于软件需求，而是以测试的观点根据软件需求整理出的一个 checklist(检查表)，作为测试该软件的主要工作内容。因此，需求采集的过程实际上是将软件开发需求中的那些具有可测试性的需求或特性提取出来，形成原始测试需求。

所谓可测试性，是指这些提取的需求或特性必须存在一个可以明确预知的结果，可以用某种方法对这个明确的结果进行判断、验证，验证是否符合文档中的要求。

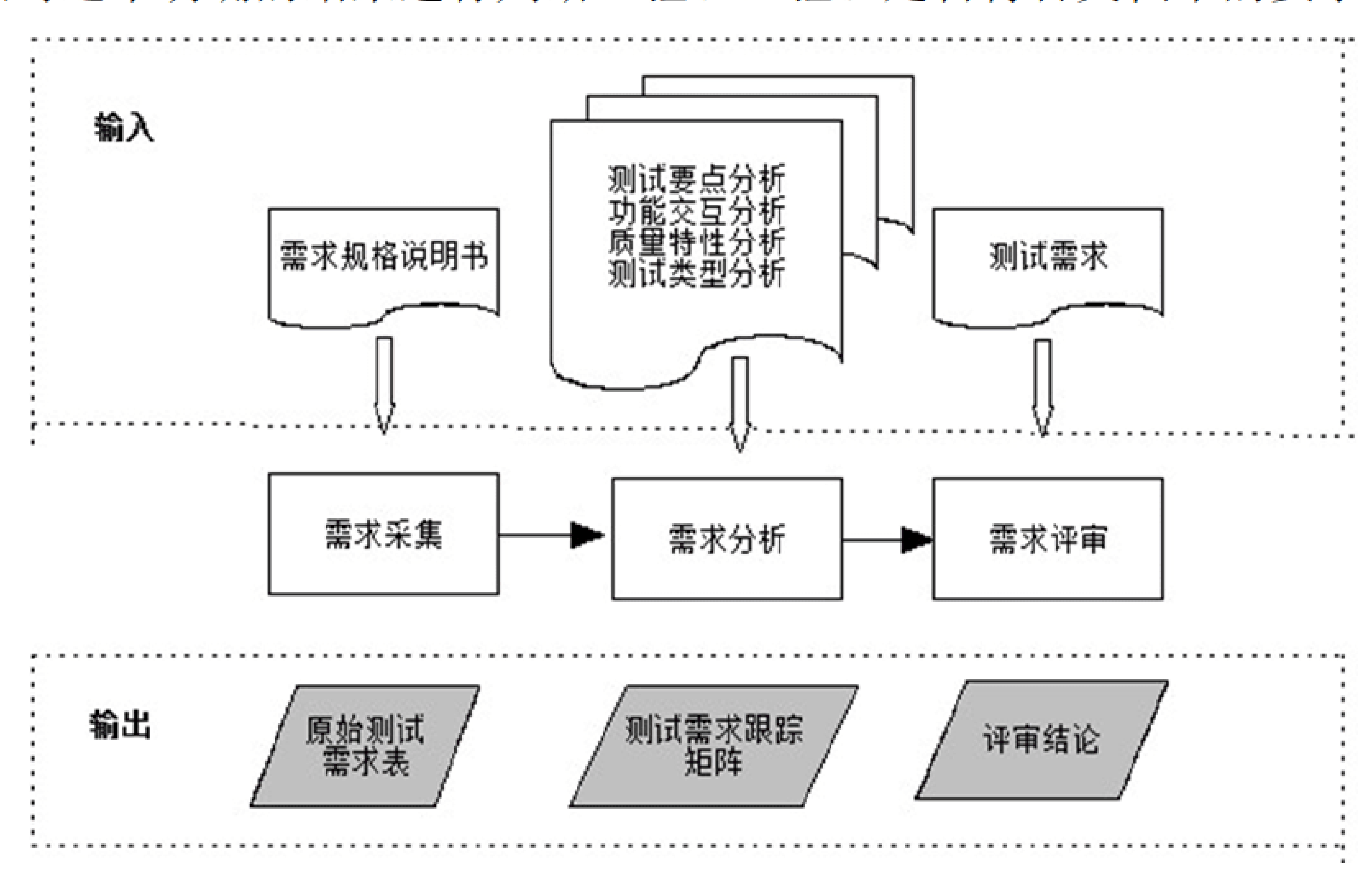


图 6-6　软件测试需求分析过程

测试需求主要通过以下途径来收集：

(1) 与被测软件相关的各种文档资料，如软件需求规格、Use Case(用例)、界面设计、项目会议或与客户沟通时有关需求信息的会议记录、其他技术文档等。在此过程中，可通过列表的形式对软件开发需求进行梳理，形成原始测试需求列表(列表的内容包括需求标识、原始测试需求描述、信息来源)。

(2) 与客户或系统分析员的沟通。

(3) 业务背景资料，如待测软件业务领域的知识等。

(4) 正式与非正式的培训。

(5) 其他。如果以旧系统为原型，以全新的架构方式来设计或完善软件，那么旧系统的原有功能与特性就成了最有效的测试需求收集途径。

在整个信息收集过程中，务必确保软件的功能与特性被正确理解，这要求测试需求分析人员必须具备优秀的沟通能力与表达能力。

2) 软件测试需求的分析

首先，测试需求分析需要考虑如下几个层面的因素。

第一层：测试阶段

在系统测试阶段，需求分析更注重于技术层面，即软件是否实现了具备的功能。如果某一流程或某个角色能够执行一项功能，那么我们相信具备相同特征的业务或角色都能够执行该功能。为了避免测试执行的冗余，可不再重复测试。而在验收测试阶段，更注重于不同角色在同一功能上能否走通要求的业务流程。因此需要根据不同的业务需要测试相同的功能，以确保系统上线后不会有意外发生。但是否有必要进行这种大量的重复性质的测试，也是见仁见智的做法，要看测试管理者对测试策略与风险的平衡能力。

目前，大多数的测试都会在系统测试中完成，验收测试只是系统测试的回归。此种情况也是合理的，关键看测试周期与资源是否允许，以及各测试阶段的任务划分。

第二层：被测软件的特性

不同的软件业务背景不同，所要求的特性也不相同，测试的侧重点自然也不相同。除了需要确保要求实现的功能正确外，银行/财务软件更强调数据的精确性，网站强调服务器所能承受的压力，ERP 强调业务流程，驱动程序强调软硬件的兼容性。在做测试分析时需要根据软件的特性来选取测试类型，并将其列入测试需求中。

第三层：测试的焦点

测试的焦点是指根据所测的功能点进行分析、分解，从而得出的着重于某一方面的测试，如界面、业务流、模块化、数据、输入域等。目前关于各个焦点的测试也有不少的指南，那些已经是很好的测试需求参考了，在此仅列出业务流的测试分析方法。

3) 软件测试需求分析举例

任何一套软件都会有一定的业务流程，也就是用户用该软件来实现自己实际业务的一个流程。简单来说，在做测试需求分析时需要列出如下业务流程：常用的或规定的业务流程、各业务流程分支的遍历、明确规定不可使用的业务流程、没有明确规定但是应该不可以执行的业务流程、其他异常或不符合规定的操作等类别。

然后根据软件需求理出业务的常规逻辑，按照以上类别提出的思路，一项一项列出各种可能的测试场景，同时借助软件的需求以及其他信息，确定该场景应该导致的结果，便形成了软件业务流程的基本测试需求。

在做完以上步骤之后，将业务流程中涉及的各种结果以及中间流程分支回顾一遍，确定是否还有其他场景可能导致这些结果，以及各中间流程之间的交互可能产生的新的流程，从而进一步补充与完善测试需求。

另外，在测试需求分析过程中，要确定测试需求的优先级别。这有利于测试工作有的放矢地展开，使测试人员清晰了解核心的功能、特性与流程有哪些，客户最为关注的是什么，由此可确定测试的工作重点在何处，更方便处理测试进度发生问题时，实现不同优先级别的功能、模块、系统等的迭代递交或取舍，从而缓和测试风险。

通常，需求管理规范的客户，会规定用户需求/软件需求的优先级别，测试需求的优先级可根据其直接定义。如果没有规定项目需求的优先级，可与客户沟通，确定哪些功能或特性是需要重点关注的，从而确定测试需求的优先级。

最后，测试需求分析对测试需求的覆盖率是有要求的。测试需求的覆盖率通常由与软件需求所建立的对应关系来确定。如果一个软件的需求已经跟测试需求存在一对一或一对多的对应关系，可以说测试需求已经覆盖了该功能点，以此类推。如果确定所有的软件需求都建立了对应的测试需求，那么测试需求的覆盖率便是测试需求覆盖点/软件需求功能点=100%，但并不意味着测试需求的覆盖程度高。因为测试需求的覆盖率只计算了显性的因素(即被明确规定的功能与特性)，而隐性的因素(即没有被明确规定但是有可能或不应该拥有的功能与特性)并未计算在内。因此，根据不断完善或实际测试中发生的缺陷，可以对测试需求进行补充或优化，并更新到测试用例中，以此来提高测试需求的覆盖程度。

4) 软件测试需求的评审

软件测试需求的评审包括如下内容：

(1) 完整性审查：应保证测试需求能充分覆盖软件需求的各种特征，重点关注功能要求、数据定义、接口定义、性能要求、安全性要求、可靠性要求、系统约束等方面，同时还应关注是否覆盖开发人员遗漏的、系统隐含的需求。

(2) 准确性审查：应保证所描述的内容能够得到相关各方的一致理解，各项测试需求之间没有矛盾和冲突，各项测试需求在详尽程度上保持一致，每一项测试需求都可以作为测试用例的设计依据。

一般采用如下形式：

(1) 相互评审、交叉评审：甲和乙在一个项目组，处在一个领域，但工作内容不同，甲的工作成果交给乙审查，乙的工作成果交给甲审查。相互评审是最不正式的一种评审形式，但应用方便、有效。

(2) 轮查：又称分配审查法，是一种异步评审方式。作者将需要评审的内容发送给各位评审员，并收集他们的反馈意见。

(3) 走查：作者将测试需求在现场向一组同事介绍，以收集大家的意见。希望参与评审的其他同事可以发现其中的错误，并能进行现场讨论。这种形式介于正式和非正式之间。

(4) 小组评审：通过正式的小组会议完成评审工作，是有计划的和结构化的评审方式。评审定义了评审会议中的各种角色和相应的责任，所有参与者在评审会议的前几天就拿到了评审材料，并对这些材料进行了独立研究。

(5) 审查：审查和小组评审相似，但更为严格，是最系统化、最严密的评审形式，包含制定计划、准备和组织会议、跟踪和分析审查结果等。

评审由如下人员组成：

(1) 在正式评审小组中，一般存在多种角色，包括协调人、作者、评审员等。

(2) 评审员需要精心挑选，保证不同类型的人员都要参与进来，通常包括开发经理、项目经理、测试经理、系统分析人员、相关开发人员和测试人员等。

测试需求分析结束后，还有一件很重要的工作，就是制定测试策略。测试策略描述当前测试的目标和所采用的测试方法。其中，当前测试的目标是针对某个应用软件系统或程序的。具体的测试任务目标是测试要达到的预期结果有哪些，在规定的时间内哪些测试内容要完成，软件产品的特性或质量在哪些方面得到确认等。测试策略还要描述测试不同阶段(单元测试、集成测试、系统测试)的测试对象、范围和方法以及每个阶段内所要进行的

测试类型(功能测试、性能测试、压力测试等)。在制定测试策略前，要确定测试策略项，测试策略包括以下几个方面：

(1) 要使用的测试技术和工具，如 60%用工具自动测试，40%手工测试。

(2) 测试完成的标准，用以计划和实施测试，以及通报测试结果，如 95%测试用例通过并且重要级别的缺陷全部解决。

(3) 影响资源分配的特殊考虑，例如有些测试必须在周末进行，有些测试必须通过远程环境执行，有些测试需要考虑外部接口或硬件接口。

在确认测试方法时，要根据实际情况，结合测试策略的特点来选择合适的方法：

(1) 根据是否需要执行被测软件来划分，有静态测试和动态测试。静态测试，如规格说明书、程序代码的审查，在工作中容易被忽视，在测试策略上应说明如何加强这些环节。

(2) 根据是否针对系统的内部结构和具体实现算法来划分，有“白盒测试”和“黑盒测试”。如何将“白盒测试”和“黑盒测试”有机地结合起来测试，也是测试策略要处理的问题之一。尽管用户更倾向于基于程序规格说明的功能测试，但是“白盒测试”能发现潜在的逻辑错误，而这种错误往往是功能测试发现不了的。

综合起来，可能要在基于测试技术的测试策略和基于测试方案的综合测试策略之间做出选择。

6.2.3　软件测试设计和开发

当测试计划完成之后，测试过程就要进入软件测试设计和开发阶段。软件测试设计建立在测试计划书的基础上，认真理解测试计划的测试大纲、测试内容及测试的通过准则，通过测试用例来完成测试内容与程序逻辑的转换，作为测试实施的依据，以实现所确定的测试目标。软件设计是将软件需求转换成为软件表示的过程，主要描绘出系统结构、详细的处理过程和数据库模式；软件测试设计则是将测试需求转换成测试用例的过程，主要描述测试环境、测试执行的范围、层次和用户的使用场景以及测试输入和预期的测试输出等。所以，软件测试设计和开发是软件测试过程中一个技术深、要求高的关键阶段。

1. 软件测试设计与开发的主要内容

软件测试设计和开发的主要内容有：

(1) 制定测试的技术方案，确认各个测试阶段要采用的测试技术、测试环境和平台，以及选择什么样的测试工具。系统测试中的安全性、可靠性、稳定性、有效性等的测试技术方案是这部分工作内容的重点。

(2) 设计测试用例，根据产品需求分析、系统技术设计等规格说明书，在测试的技术方案基础上，设计具体的测试用例。

(3) 设计测试用例特定的集合，满足一些特定的测试目的和任务，即根据测试目标、测试用例的特性和属性(优先级、层次、模块等)，选择不同的测试用例，构成执行某个特定测试任务的测试用例集合(组)，如基本测试用例组、例外测试用例组、性能测试用例组、完全测试用例组等。

(4) 测试开发：根据所选择的测试工具，将所有可以进行自动化测试的测试用例转换为测试脚本的过程。

(5) 测试环境的设计，根据所选择的测试平台以及测试用例所要求的特定环境，进行服务器、网络等测试环境的设计。

在软件测试设计中，需要考虑的要点有：①所设计的测试技术方案是否可行、是否有效、是否能达到预期的测试目标；②所设计的测试用例是否完整、边界条件是否考虑、覆盖率能达到多高；③所设计的测试环境是否和用户的实际使用环境比较接近。

其关键是做好测试设计前的知识传递，将设计/开发人员已经掌握的技术、产品、设计等知识传递给测试人员；同时，要做好测试用例的审查工作，不仅要通过测试人员的审查，还要通过设计/开发人员的审查。

在软件测试设计和开发阶段，按国家标准 GB/T 9386-1988《计算机软件测试文件编制规范》的要求，我们要编写《测试设计说明》、《测试用例说明》、《测试规程说明》、《测试项传递报告》等文档。

2. 测试用例的设计方法和管理

测试设计工作中的一项重要工作就是测试的准备工作。一般来讲，由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。测试准备工作中的重要内容就是设计测试用例。

软件测试用例的设计方法有白盒测试和黑盒测试相对应的设计方法，黑盒测试的用例设计，采用等价类划分、因果图法、边值分析、用户界面测试、配置测试、安装选项验证等方法，适用于功能测试和验收测试。白盒测试的用例设计有以下方法：

(1) 采用逻辑覆盖(包括程序代码的语句覆盖、条件覆盖、分支覆盖)的结构测试用例的设计方法。

(2) 基于程序结构的域测试用例设计方法。“域”是指程序的输入空间，域测试是在分析输入空间的基础上，完成域的分类、定义和验证，从而对各种不同的域，选择适当的测试点(用例)进行测试。

(3) 数据流测试用例的设计方法，是通过程序的控制流，从建立的数据目标状态的序列中发现异常的结构测试方法。

(4) 根据对象状态或等待状态变化来设计测试用例，也是比较常见的方法。

(5) 基于程序错误的变异来设计测试用例，可以有效地发现程序中某些特定的错误。

(6) 基于代数运算符号的测试用例设计方法，受分支问题、二义性问题和大程序问题的困扰，使用较少。

测试用例要经过创建、修改和不断改善的过程，测试用例具有以下属性：

(1) 测试用例的优先级次序，优先级越高，被执行的时间越早，执行的频率越高。由最高优先级的测试用例组构成基本验证测试，每次构建软件包时，都要被执行一遍。

(2) 测试用例的目标性，有的测试用例为主要功能而设计，有的测试用例为次要功能而设计，有的则为系统的负载而设计，有的则为一些特殊场合而设计。因此，需要根据不同的目标设计不同的测试用例。

(3) 测试用例所属的范围，属于哪一个组件或模块，这种属性被用来管理测试用例。

(4) 测试用例的关联性，测试用例一般和软件产品的特性相联系，多数情况下验证某个产品的功能。这种属性可被用于验证被修改的软件缺陷，或对软件产品紧急补丁包的测试。

(5) 测试用例的阶段性，属单元测试、集成测试、系统测试、验收测试中的某个阶段。这样对于每个阶段，构造测试用例的集合并被执行，容易计算出该阶段的测试覆盖率。

(6) 测试用例的状态性，当前是否有效，如果无效，被置于 inactive 状态，不会被执行，只有被激活的(active)测试用例才被执行。

(7) 测试用例的时效性，针对同样的功能，可能所用的测试用例不同，因为不同的产品版本在产品功能、特性等方面的要求不同。

(8) 所有者、日期等特性，测试用例还包括由谁、在什么时间创建，又由谁、在什么时间修改。

根据上述特性，再结合测试用例的编号、标题、描述(条件、步骤、期望结果)等，就可以对测试用例进行基于数据库方式的良好管理。测试用例设计完之后，要经过非正式和正式的审查：非正式的审查一般在 QA 或测试小组(部门)内部进行，包括同测试组人员互相检查，或让资深人员、测试组长帮助审查；正式的审查一般通过正式文档将已设计好的测试用例分发给相应的系统分析人员、设计人员和程序员，让他们先通读一遍，将发现的问题记下来。然后由测试组长或项目经理召开测试用例审查会，由测试设计人员先对测试用例的设计思想、方法、思路等进行说明，然后由系统分析人员、设计人员和程序员把问题提出来，测试人员回答，必要时做些讨论。

审查完的测试用例，经修改后，就可以直接用于手工测试或用于测试脚本的开发。

3. 测试开发

根据所选的测试工具脚本语言，如 HP UFT VBScript、IBM Rational SQA Basic，编写测试脚本，将所有可以进行自动化测试的测试用例转换为测试脚本。其输入就是基于测试需求的测试用例，输出是测试脚本和与之对应的期望结果，这种期望结果一般存储在数据库中或特定的格式化文件中。

1) 测试开发的步骤

首先要设立测试脚本开发环境，安装测试工具软件，设置管理服务器和具有代理的客户端池，建立项目的共享路径、目录，并能连接到脚本存储库和被测软件等。

然后录制测试初始化过程、独立模块过程、导航过程和其他操作过程，结合已经建立的测试用例，对录制的测试脚本进行组织、调试和修改，构造有效的测试脚本体系，并建立外部数据集合。

由于被测系统处在不完善阶段，在运行测试脚本的过程中，容易中断，因此以在测试脚本开发时，既要处理好这种错误，及时记录当时的状态，又要能继续执行下去。处理这个问题时有一些解决办法，如跳转到别的测试过程，调用一个能够清除错误的过程等。

2) 测试开发常见的问题

测试开发很乱，与测试需求或测试策略没有对应性；测试过程不可重用；测试过程被作为编程任务来执行，导致脚本可移植性差。这些问题应该避免，在脚本的结构、模块化、参数传递、基础函数等方面设计好。

6.2.4　软件测试的执行

当测试用例的设计和测试脚本的开发完成之后，就开始执行测试。测试的执行有手工测试和自动化测试两种方式。手工测试在合适的测试环境中，按照测试用例的条件、步骤要求，准备测试数据，对系统进行操作，比较实际结果和测试用例所描述的期望结果，以确定系统是否正常运行或正常表现；自动化测试是通过测试工具，运行测试脚本，得到测试结果。自动化测试的管理相对比较容易，测试工具不会做小动作，会不打折扣地执行测试脚本，并能自动记录下测试结果。

在测试执行阶段，测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书以及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，作好测试前的准备工作。

在本阶段，测试人员要编写《测试日志》、《测试事件报告》。

1. 测试阶段目标的检查

要对每个测试阶段(代码审查、单元测试、集成测试、功能测试、系统测试和验收测试、安装测试等)的结果进行分析，保证每个阶段的测试任务得到执行，达到阶段性目标。

1) 代码审查

不是指编程人员互查，而是指测试人员参与的代码会审。代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。会审小组由组长、两或三名程序设计和测试人员及程序员组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件的基础上，召开代码会审会议，程序员逐句讲解程序的逻辑，并展开热烈的讨论甚至争议，以揭示错误的关键所在。实践表明，程序员在讲解过程中能发现许多自己原来没有发现的错误，而讨论和争议则进一步促使了问题的暴露。

2) 单元测试

目的在于发现各模块内部可能存在的各种差错。

单元测试集中检查软件设计的最小单位——模块，通过测试发现实现模块的实际功能与定义模块的功能说明不符合的情况，以及编码错误。由于模块规模小、功能单一、逻辑简单，测试人员有可能通过模块说明书和源程序，清楚地了解模块的 I/O 条件和模块的逻辑结构，采用结构测试(白盒法)的用例，尽可能达到彻底测试，然后辅之以功能测试(黑盒法)的用例，使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

单元测试一般由程序员自己做，但必须提交单元测试用例和测试报告，测试人员要审查单元测试用例和测试报告。

3) 集成测试

集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。例如：数据穿过接口时可能丢失；一个模块与另一个模块集成可能相互间造成有害影响；把子功能组合起来可能不产生预期的主要功能；个别看起来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有错误等。

测试人员为发现与外部接口及内部参数传递等方面的问题，要抓住关键模块，关键模

块应尽早测试，并将自顶向下、自底向上两种测试策略结合起来，对各个模块严格执行。由于设计系统不同的模块、不同的层次或不同的部门，容易造成一些漏洞、疏忽，要根据设计文档多提问题、集体审查。

4）功能测试

目的是向未来的用户表明系统能够按预定要求的功能那样工作，这时的测试是直接操作完整的软件系统，需要站在用户的角度，尽量模拟用户使用的各种情景，甚至让用户参与测试。

5）系统测试

目标是保证系统在实际的环境中能够稳定、可靠地运行下去，包括恢复性测试、安全性测试、强度测试和性能测试等。系统测试技术要求高，占用资源比较多，所以应充分设计好、准备充分。

6）验收测试

验收测试既可以是非正式的测试，也可以是正式的、有计划的测试。其目的是向未来的用户表明系统能够像预定要求那样工作。一个软件产品可能拥有众多用户，不可能由每个用户验收，此时多采用称为α、β测试的过程。α测试是指软件开发公司组织内部人员模拟各类用户对即将发布的软件产品进行测试，试图发现错误并修正。α测试的关键在于尽可能逼真地模拟实际运行环境和可能的用户操作方式。经过α测试调整的软件产品称为β版本。紧随其后的β测试是指组织公司外部的典型用户试用β版本，并要求用户报告异常情况、提出批评意见，然后再对β版本进行改错和完善。

经历完上述测试过程，对软件进行测试后，软件基本满足开发的要求，测试宣告结束，经验收后，将软件提交用户。

2. 跟踪测试用例的执行

测试用例的执行直接关系到测试的效率、结果，不仅要做到测试效率高，而且要保证结果正确、准确、完整等，关键首先是提高测试人员的素质和责任心，树立良好的质量文化意识，其次要通过一定的跟踪手段从某些方面保证测试执行的质量。

测试效率的跟踪比较容易，按照测试任务和测试周期，可以得到期望的曲线，然后每天检查测试结果，了解是否按预期进度进行。图 6-7 所示为测试执行情况的跟踪曲线。

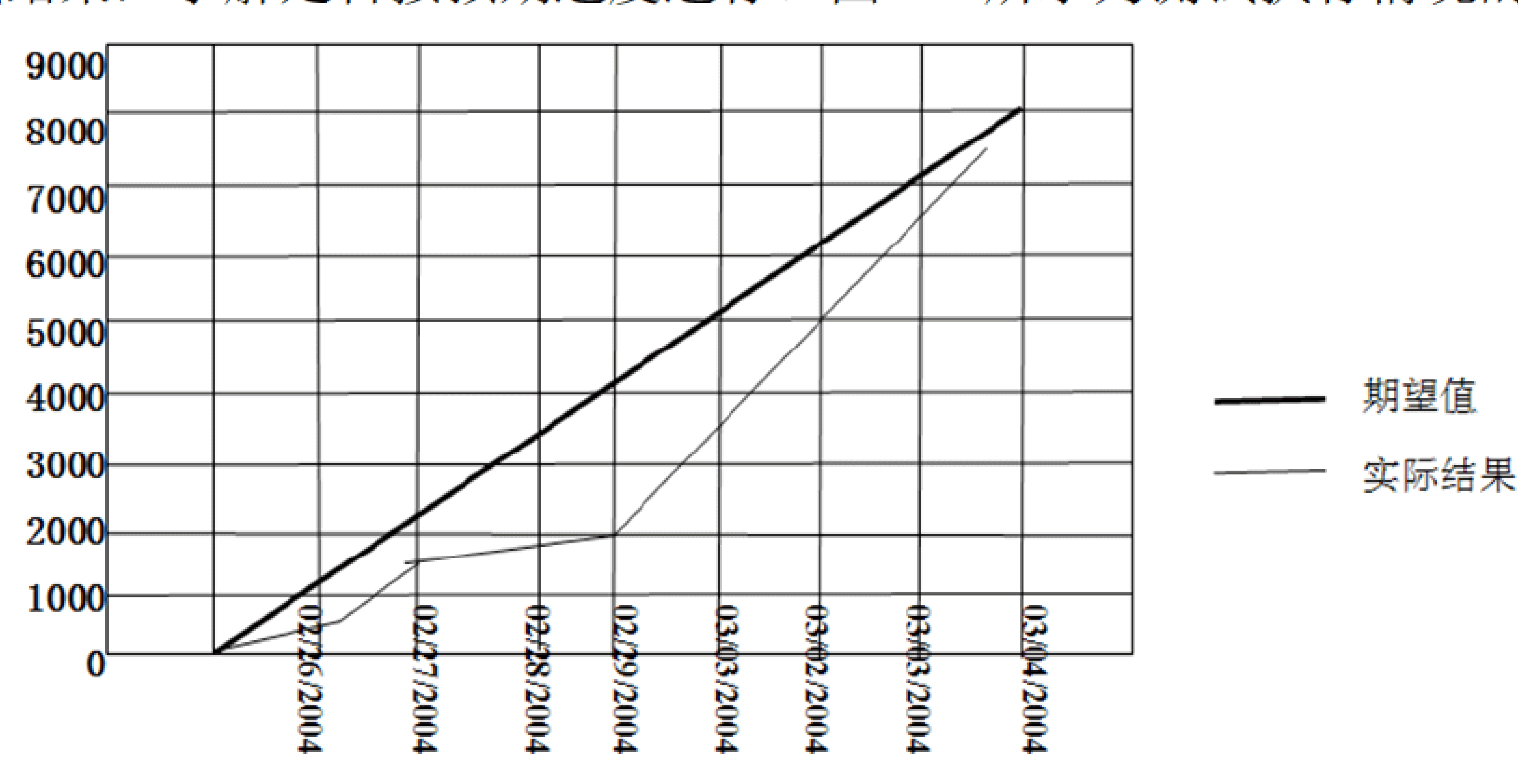


图 6-7　测试执行情况的跟踪曲线

测试结果的跟踪相对存在一些风险，但如果记录好每个人的执行测试情况，即知道哪个测试用例是谁执行的。一旦某个 bug 漏掉，就可以追溯到具体责任人。

3. bug 的跟踪和管理

在测试过程中发现的软件错误或缺陷，可提交或纳入到软件的缺陷管理过程中。bug 的跟踪和管理一般由数据库系统来执行，但数据库系统也依赖于一定的规则和流程来进行，主要的思路有：①设计好每个 bug 应该包含的信息条目、状态分类等；②通过系统自动发出邮件给相应的开发人员和测试人员，使得任何 bug 都不会错过，并能得到及时处理；③通过日报、周报等各类项目报告来跟踪目前 bug 状态；④在各个大小里程碑之前，召开有关人员参与的会议，对 bug 进行会审；⑤通过一些历史曲线、统计曲线等进行分析，预测未来情况。

4. 和项目组外部人员的沟通

为了使测试进展顺利，和项目组外部人员的良好沟通是必要的，这能使测试碰到的问题比较容易解决，测试中发现的 bug 的处理效率也会提高。为此推荐一些方法：①通过一种合适的、可接受的方式指出对方的问题，尽量做到对事不对人；②每周要有一次不同部门参加的会议；③建立大项目的邮件组，包含各部门主要人员的邮件地址；④在同一大项目组的开发人员及测试人员的日报、周报等要互相抄送；⑤适当搞些类似于 Party 的活动，改善关系，增加了解。

5. 测试执行结束和测试总结

测试执行全部完成，并不意味着测试项目的结束，测试项目结束的阶段性标志是将测试报告或质量报告发出去之后，能得到测试经理或项目经理的认可。除了测试报告或质量报告的写作之外，还要对测试计划、设计和执行等进行检查、分析，完成项目的总结，编写《测试总结报告》。通常包括以下活动：

(1) 审查测试全过程：在原来跟踪的基础上，要对测试项目全过程、全方位地审视一遍，检查测试计划、测试用例是否得到执行，检查测试是否有漏洞。

(2) 对当前状态的审查：包括产品 bug 和过程中没解决的各类问题。对产品目前存在的缺陷逐个进行分析，了解对产品质量影响的程度，从而决定产品的测试能否告一段落。

(3) 结束标志：根据上述两项的审查进行评估，如果所有测试内容完成、测试的覆盖率达到要求以及产品质量达到已定义的标准，就可以定稿测试报告，并发送出去。

(4) 项目总结：通过对项目中的问题加以分析，找出流程、技术或管理中存在问题的根源，避免今后发生，并获得项目成功经验。

6.2.5　软件测试文档

软件测试是一个很复杂的过程，特别是随着测试过程越来越完善，产生的各种测试过程文档也越来越多，这些文档都与相关的测试活动有紧密的联系，它们对于保证软件的质量及其运行有着重要意义。因此，我们必须把对它们的要求、过程及测试结果以正式的文档形式写出，并将它们放置在组织的软件过程改进库和测试配置库中。测试文档的编写是测试工作规范化的组成部分，是整个测试过程管理体系具体实施过程的重要内容。

1. 软件测试文档的概念

软件测试文档描述要执行的软件测试及测试结果，用来记录、描述、展示测试过程中一系列测试信息的处理过程，通过书面或图示的形式对软件测试过程中的活动或结果进行描述、定义及报告，记载了整个测试的过程和成果。测试文档不只在测试阶段才考虑，而在软件开发的需求分析阶段就开始着手，因为测试文档与用户有着密切的关系。在设计阶段的一些设计方案也应在测试文档中得到反映，以利于设计的检验。测试文档对于测试阶段工作的指导与评价作用更是非常明显。需要特别指出的是，在已开发的软件投入运行的维护阶段，常常还要进行再测试或回归测试，这时仍要用到测试文档。

作为测试人员，在测试过程中应将各种标准测试文档提交给测试项目组，以确保软件测试项目的质量。也就是说，测试人员的工作绩效与文档的高质量提交也是息息相关的，它描述项目测试过程的每一个细节。因此，从某种程度上讲，测试管理的核心内容包括测试文档管理。

从内容上说，软件测试文档大致可以分为测试成果文档和测试过程文档两大类。测试成果文档作为项目可交付物的组成部分，重要性自然不言而喻。测试过程文档主要记录了项目测试过程中的各种信息，为测试人员提供决策依据，以保证项目的顺利实现。另一方面，测试过程文档也是测试过程最为宝贵的资产，通过对过程文档进行归纳和分析，可以对测试项目的成功经验和失败教训了然于胸，从而使后续的测试运作更加有的放矢。

测试项目的阶段性成果是以测试文档的形式体现的，测试项目的实施在某种程度上是由测试文档驱动的。从测试文档的角度来看，软件测试过程就是一个文档制作与执行的过程。在软件测试的过程中，每项工作的事前计划、事中测试记录、事后分析结果都要形成相应的测试文档，文档包括与项目相关的资源及其使用情况。

因此，测试文档是软件项目的一部分，没有正式的测试文档的活动，就不是规范的测试。测试文档的编制和管理在软件测试中占有突出的地位和相当大的工作量，高质量地编制、变更、修正、管理和维护文档，对于提高项目测试的质量和客户满意度有着重要的现实意义。

2. 软件测试文档的作用

软件测试项目是否高质量完成，一般可以从两个方面进行评价：①能否提供高质量的测试活动和结果；②能否提供有效的测试文档。对于后者，高质量的测试文档就是前者是否高质量完成的证明。

1) 提高软件测试过程的能见度

标准规范、齐全的文档会详细记录测试过程中发生的事件，便于测试人员掌握测试进度、测试质量以及各种资源的调配。同时，文档有助于测试人员与开发人员明确了解各自的职责，信息互通，共同把握测试和开发的节奏。

2) 文档化能规范测试问题的反馈，提高测试效率

测试人员用一定时间编制、整理测试文档，可以使测试人员对各个阶段的工作都进行周密思考和理顺，找出存在的问题，从而减少差错，提高项目测试质量。例如，测试过程中肯定会遇到各种各样的问题，诸如软件问题或测试设置等需要向开发组反馈来寻求解决，

通过对文档的检查，在项目测试早期发现文档错误和不一致，加以及时纠正，可以减少因深入项目而导致的大问题的出现以及为纠正失误而付出更大的成本。

这类问题又分两种情况：①重要的反馈迟迟得不到解决和回复，当文档化做得好时，在出现问题的时候，打开文档可以一目了然，责任没法推卸。卡耐基曾说过，通常人是不会或没有人愿意承认自己是错的，即使承认，也并不是百分百这么认为，我们在测试过程中就要避免这样的问题发生；②有些问题在不同部门和不同的阶段频繁出现，简单而又琐碎的重复问题会让测试人员疲于奔命，效率低下。这时，文档化的常见问题集 FAQ 对项目测试就显得意义重大。

3) 便于团队成员之间的交流与合作

描述清楚、完备的测试文档便于项目组领导了解测试过程中的各项指标，为开发团队与测试团队之间架起一座桥梁。文档是一种无声的语言，记录了项目测试过程中有关测试配置、测试运行、测试结果等方面的信息，有利于项目管理人员、测试人员之间的交流和合作。

另一方面，测试文档的重要性还表现在对项目传承的重要性，有了好的文档，当项目有新成员进入时，测试文档就可以承担起指导新成员快速工作的作用，而不是单单询问原来的成员，节省大家的时间。还有，当测试完成后，测试文档就将成为项目测试的文字载体，在后续人员培训方面提供详尽的素材。

4) 测试文档是测试人员经验提升的最好途径

善于学习，对于任何职业而言都是前进所必需的动力。对于软件测试来说，这种要求就更高了，项目文档对于项目测试人员的素质提升大有裨益。目前不少企业在进行项目测试时都会出现一个通病：由于人员素质有限，许多的决定只凭口头叙述，缺少足够的文字记录，以至于出现问题时往往显得无所适从。从本质上讲，测试文档强调的是一种规范化管理，要求项目人员利用书面语言进行沟通表达，以指引项目运作。

当然，测试人员不应该只为写测试文档而写文档，良好的文档是思想交流、沟通的基础，也是整理和理清思路的基础。不懂得从经验中学习和成长，永远不会有质的提高。只有在每次完成一个测试任务后，都有目的地总结，找到自己的不足，一名合格的测试人员才可能成长起来。

5) 有利于项目测试的监控作用

测试本身是一项风险很高的工程，需要进行严谨的项目监控。阶段性的检查、评审和文档化成果是重要的方法之一，详尽而规范的测试文档成果不仅有利于监控项目进度，也有利于项目验收。

6) 有利于测试工作的开展

软件测试文档对测试工作的开展有诸多帮助，比如：

(1) 验证需求的正确性。测试文档中规定了用以验证软件需求的测试条件，研究这些测试条件对弄清用户需求的意图是十分有益的。

(2) 检验测试资源。测试计划不仅要用文件的形式把测试过程规定下来，还应说明测试工作必不可少的资源，进而检验这些资源是否可以得到，即它的可用性如何。如果某个测试计划已经编写出来，但所需资源仍未落实，那就必须及早解决。

(3) 明确任务的风险：有了测试计划，就可以弄清楚测试可以做什么，不能做什么。了解测试任务的风险有助于对潜伏的可能出现的问题事先作好思想上和物质上的准备。

(4) 生成测试用例：测试用例的好坏决定着测试工作的效率，选择合适的测试用例是做好测试工作的关键。在测试文档编制过程中，按要求精心设计测试用例有重要的意义。

(5) 评价测试结果：测试文档包括测试用例，即若干测试数据及对应的预期测试结果。完成测试后，对测试结果与预期结果进行比较，便可对已进行的测试提出评价意见。

(6) 再测试：测试文档规定和说明的内容在维护阶段由于各种原因的需求而进行再测试时，非常有用。

(7) 决定测试的有效性：完成测试后，把测试结果写入文件，这为分析测试的有效性，甚至整个软件的可用性提供了依据。同时还可以证实有关方面的结论。

3. 测试文档常见问题

测试文档是否专业已成为测试管理和测试人员的重要评价指标之一。但是，普遍还会存在以下缺点：

1) 文档编写不够规范

主要是测试文档的内容描写不够完善，在编写各种测试文档的过程中，虽然大家都按事先规定的模式进行了编写，但编写的内容经常不够完善。要么文档极其简单，相当于没有文档；要么文档流于形式，没有什么实际的价值，甚至有的测试文档与测试过程完全不符。

2) 测试文档没有统一入库管理

随着软件开发的不断深入、升级，新 bug 不断产生，各种测试文档越来越多，没有建立测试文档资料库。在测试过程中没有对每一个阶段的文档进行整理、分层次管理，各类文档资料缺少一致性。不同时期的各种测试文档零散存在，造成查询测试文档时非常困难。在众多的测试文档中，其中一些文档必定是关键文档，起到非常重要的作用，对于这类测试文档没有设定优先级别特别说明。

3) 只重视测试文档的形式，实用性不强

在实际的测试过程中，编制人员没有时间去关心它们的用途，也不知道哪些部门在使用，更多的是在规定的时间内完成任务，以免影响考核成绩。这样一来，一些不实用的、重复的文档不但阻碍着测试的执行效率，而且影响项目的整体进度。因此，文档的制定要实用，以减少繁文缛节的文字工作。

4. 测试文档的管理

从前面我们看到，测试文档对项目管理的作用是不容置疑的，但测试文档的管理却又通常是项目管理中最容易忽略的。通常在测试文档管理中应该注意以下几个方面：

1) 建立测试文档管理制度

重点应体现为以下两点：①要对测试文档的名称、标识、类型、责任人、内容等基本内容做出事先安排，给出测试文档总览表；②制定各种测试文档的管理程序，如批准、发布、修订、标识、贮存、传递、查阅等，为测试文档的配置管理铺设良好的基础平台。

2) 文档版本管理，而且非常重要

版本混乱是测试文档的致命伤，测试文档的有效管理必须实行版本控制。

3) 创建测试文档库的访问规则，这是文档管理的重要环节

访问规则确定谁可以访问、阅读、升级以及在文档库中添加文档。同时，文档库还应定期进行检查，以便对哪些文件进行存档或对哪些旧文件进行清理，以确保文档管理符合项目测试组的需求。

4) 使用工具管理文档

对于大型的项目测试，在整个测试周期中都会有大量的文档。测试文档内容也是在不断变化的，有的是连续的、承前启后的，有的是新增加的，也有的是废除的。这可能需要一个统一的文档管理工具，分门别类统一存放管理各种测试文档。

总之，测试文档在软件测试过程中起到关键的作用，从某种意义上来说，测试文档是项目测试规范的体现和指南，按照规范要求编制一整套测试文档的过程，就是完成测试项目的过程。

5. 测试文档与测试过程的关系

GB/T 9386-2008《计算机软件测试文档编制规范》对测试文档与测试过程的关系有明确的图形表示，见图 6-8。

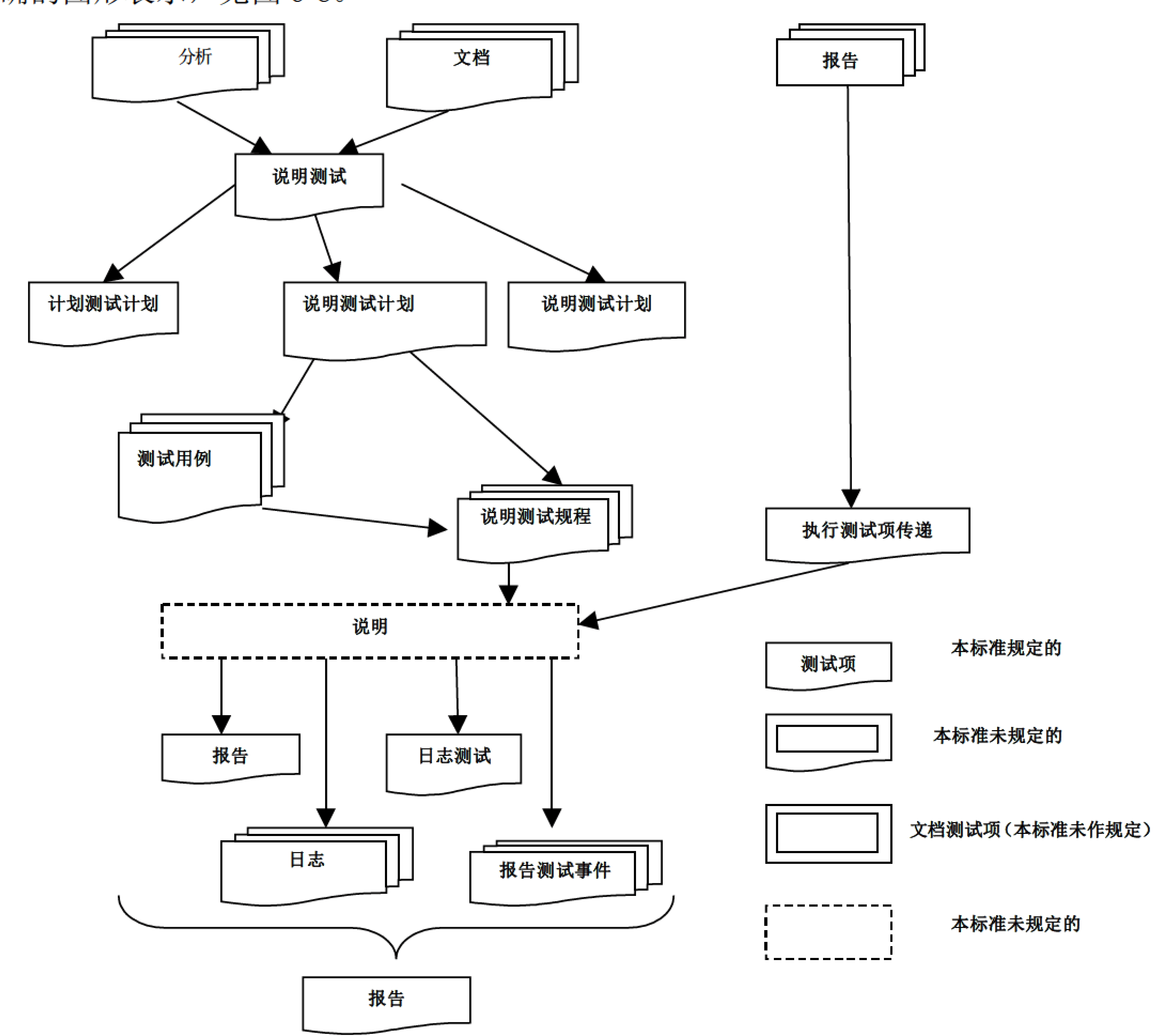


图 6-8　测试文档与测试过程的关系图

6.2.6　软件测试用例、测试数据与测试脚本

测试用例是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某条程序路径或核实是否满足某个特定需求。它构成了设计和制定测试过程的基础，决定着测试设计和测试开发的类型以及所需的资源。

测试数据是在测试中使用的实际值(集合)或执行测试需要的元素。测试数据创建要测试的条件(作为输入或预先存在的数据)，并且用于核实特定的用例或需求是否已经成功得到实施(将实际结果和预期结果相比较)。

测试脚本是自动执行测试过程(或部分测试过程)的计算机可读指令。测试脚本可以被创建(记录)或使用测试自动化工具自动生成，或用编程语言编程来完成，也可综合前三种方法来完成。测试脚本的目的在于以高效率和有效的方式来实施和执行测试过程。

1. 测试用例

测试用例的设计和编制是软件测试活动中最重要的活动之一。

1) 测试用例的作用

测试用例在软件测试中主要有如下作用：

(1) 指导测试的实施

测试用例主要适用于集成测试、系统测试和回归测试。在实施测试时测试用例作为测试的标准，测试人员一定要按照测试用例严格按用例项目和测试步骤逐一实施测试，并将测试情况记录在测试用例管理软件中，以便自动生成测试结果文档。

根据测试用例的测试等级，集成测试应测试哪些用例，系统测试和回归测试又该测试哪些用例，在设计测试用例时都已作明确规定，实施测试时测试人员不能随意作变动。

(2) 规划测试数据的准备

通常情况下测试数据是与测试用例分离的。按照测试用例配套准备一组或若干组测试原始数据，以及标准测试结果。尤其像测试报表之类数据集的正确性，按照测试用例规划准备测试数据是十分必要的。

除正常数据之外，还必须根据测试用例设计大量边缘数据和错误数据。

(3) 编写测试脚本的设计规格说明书

为提高测试效率，软件测试已大力发展自动测试。自动测试的中心任务是编写测试脚本。如果说软件工程中软件编程必须有设计规格说明书，那么测试脚本的设计规格说明书就是测试用例。

(4) 评估测试结果的度量基准

完成测试实施后需要对测试结果进行评估，并且编制测试报告。判断软件测试是否完成、衡量测试质量需要一些量化的结果。例如：测试覆盖率是多少、测试合格率是多少、重要测试合格率是多少，等等。以前统计基准是软件模块或功能点，显得过于粗糙。采用测试用例作为度量基准更加准确、有效。

(5) 分析缺陷的标准

通过收集缺陷，对比测试用例和缺陷数据库，分析确证是漏测还是缺陷复现。漏测反映了测试用例的不完善，应立即补充相应测试用例，最终达到逐步完善软件质量的目的。

若已有相应测试用例，则反映实施测试或变更处理存在问题。

2) 编制测试用例的方法

通常我们是按如下方法编制测试用例的：

(1) 编写测试用例文档

编写测试用例文档应有文档模板，必须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。

软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位，形成测试用例文档，但这并不是绝对的。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息：用例编号、用例名称、测试等级、入口准则、测试输入、操作步骤、期望结果(含判断标准)、出口准则、注释等。

(2) 测试用例的设置

测试用例可以按功能设置、按路径设置以及按这两者的组合设置。

按功能设置是最简捷的，按用例规约遍历测试每一功能。

对于复杂操作的程序模块，其各功能的实施是相互影响、紧密相关、环环相扣的，可以演变出数量繁多的变化。没有严密的逻辑分析，产生遗漏是在所难免的。路径分析是一个很好的方法，其最大的优点在于可以避免漏测。

但路径分析法也有局限性。若一个子系统有十余个或更多个模块，这些模块相互有关联。再采用路径分析法，其路径数量呈几何级增长，达 5 位数或更多，就无法使用了。那么子系统模块间的测试路径或测试用例还是要靠传统方法来解决。这是按功能、路径混合模式设置用例的由来。

(3) 设计测试用例

测试用例可以分为基本事件、备选事件和异常事件。设计基本事件的用例，应该参照用例规约(或设计规格说明书)，根据关联的功能、操作按路径分析法设计测试用例；而对于孤立的功能，则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达 100%。

设计备选事件和异常事件的用例，则要复杂和困难得多。往往在设计编码阶段形成的文档对备选事件和异常事件分析描述不够详尽，而测试本身则要求验证全部非基本事件，并同时尽量发现其中的软件缺陷。

可以采用的常用软件测试基本方法：等价类划分法、边界值分析法、错误推测法、因果图法、逻辑覆盖法等。视软件的不同性质采用不同的方法。如何灵活运用各种基本方法来设计完整的测试用例，并最终实现暴露隐藏的缺陷，全凭测试设计人员的丰富经验和精心设计。

(4) 对测试用例的评审

测试用例是软件测试的准则，但并不是一经编制完成就成为准则。测试用例在设计编制过程中要组织同级互查。完成编制后应组织专家评审，获得通过才可以使用。评审委员会可由项目负责人、测试人员、编程人员、分析设计人员等有关人员组成，也可邀请客户

代表参加。

(5) 测试用例的修改更新

测试用例在形成文档后也还需要不断完善，主要来自三方面的缘故：①在测试过程中发现设计测试用例时考虑不周，需要完善；②在软件交付使用后反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成；③软件自身的新增功能以及软件版本的更新，测试用例也必须配套修改更新。

一般小的修改完善可在原测试用例文档中修改，但文档要有更改记录。软件的版本升级更新，测试用例一般也应随之编制升级更新版本。

1) 测试用例管理软件

测试用例的管理最好是借助测试用例管理软件。它的主要功能有三个：①能将测试用例文档的关键内容，如编号、名称等自动导入管理数据库，形成与测试用例文档完全对应的记录；②可供测试实施时及时输入测试情况；③最终实现自动生成测试结果文档，包含各测试度量值、测试覆盖表和测试通过或不通过的测试用例清单列表。

有了管理软件，测试人员无论是编写每日的测试工作日志，还是出软件测试报告，都会变得轻而易举。

2) 基于需求的测试用例

软件测试的需求有三个层次，即任务需求、用户需求、功能需求。测试需求分析和测试用例设计参照的是软件需求规格说明书。

测试需求的主要来源是系统需求规格说明书，但有些需求无法从需求文档中获得，可借助概要设计文档或详细设计文档获得，或直接从最终的软件产品获得。测试人员依据这些信息编写测试需求，为了提高需求分析的覆盖率，用例设计人员可通过分析软件的任务规则和工程测试经验，提出软件产品隐含的需求，以保证最终的测试需求满足测试要求。

实际上，测试用例的设计也就是测试需求细化的过程，可以说，有多细的测试需求，就能设计出多细的测试用例。通常采用等价类划分法划分有效和无效的数据集，采用边界值法找到被测软件的输入数据的边界值数据。在基于需求的测试用例设计中，这两种方法既是基础又是补充，当测试数据量比较大时，通常采用自动化测试工具或正交试验法。测试用例的内容项可依据具体情况而定，通常包含测试用例编号、测试操作步骤和预期结果等。在软件系统测试过程中，软件需求决定了测试用例设计，而测试用例设计的效果则直接决定了整个软件测试项目的成败，因此测试需求分析和测试用例设计是密不可分的，前者是后者的依据，后者是前者的体现，做好需求到测试用例的转换，才能保证整个测试项目的效果。

在软件系统测试过程中，软件测试需求决定了测试用例设计，而测试用例设计关系到测试用例的运行。应该说，设计出了什么样的测试用例，就需要针对性地选择测试用例运行方式。测试用例的运行一般采用测试者手工运行、编写驱动程序运行、借助自动化工具(如 HP UFT)运行等方式。测试用例设计的优劣直接关系着测试用例运行的工作量，编写脚本自动运行程序是解决此问题的不错方式。现阶段，编写脚本自动运行程序来驱动测试用例在用例运行时的趋势，这不仅可以节约第一次测试的工作量，而且还可以减少后续的回归测试的工作量。

2. 测试数据

在测试过程中，我们都知道需要做测试用例的设计，但其中很重要的工作是测试数据的设计。因为在测试设计活动中，需要确定和描述两个重要的工作产品：测试过程和测试用例。如果没有测试数据，这两个工作产品将无法实施和执行。它们只是对条件、场景和路径的一些说明，而没有具体的值用来简明地确定它们。测试数据虽然本身不是一个产出物，但是它对测试的成败产生重要的影响。没有测试数据，将无法实施和执行测试，尤其当要求测试数据作为创建条件的输入、作为评估需求的输出结果以及作为支持(作为测试的前置条件)的值时。因此，确定这些值是一项重要的工作，并且这项工作在确定测试用例后完成。

对于测试执行人员来说，测试用例的表示内容包括这么几个方面：测试目标、被测功能点描述、测试运行环境、执行方法(包括测试步骤、输入测试数据或测试脚本)、测试的期望结果、测试的实际结果、其他辅助说明。

从以上几点，我们可以看到输入测试数据只是设计测试用例的一个步骤，而不是全部。

1) 测试用例与测试数据的关系

测试用例与测试数据之间的关系是：①测试用例的主体部分包括测试逻辑和测试数据；②测试用例由主体部分、测试用例相关信息(说明、附件等)和跟踪、管理所需的各种内容组成；③等价类划分、边界值分析等方法主要用于测试数据的设计；④测试逻辑主要包括测试的前提条件、操作步骤和预期结果等；⑤测试逻辑主要通过场景分析来设计。

通过这种方式设计的测试用例，逻辑和数据分离，用例逻辑清晰、内容简洁且易于理解，也有利于转换成自动化测试脚本。

2) 测试数据的属性

确定实际的测试数据时，需要说明处理测试数据的四个属性：深度、宽度、范围及构架。

深度是在测试中所使用数据的容量或数量。深度是一个需要考虑的重要事项，因为数据太少可能无法反映现实生活中的条件，而数据太多将难以管理和维护。理想条件下，测试应首先使用一个小的支持关键测试用例的数据集(通常为正面测试用例)。随着测试过程中信心不断增强，应该增加测试数据，直到数据深度完全体现出部署环境(或适当可行的范围)为止。

测试数据的深度与用作输入和用于支持测试(在预先存在的数据中)的测试数据相关。

宽度是指测试数据值变化的程度。创建更多的记录就可以增加测试数据的深度。虽然这通常是一个好的解决方法，但是它无法解决我们期望在实际数据中看到的数据真实变化的问题。如果在测试数据中没有这些变化，我们可能无法确定缺陷。

范围是测试数据与测试目标之间的关联关系，它和测试深度和测试宽度相关。具有许多数据并不意味着其数据一定正确。与处理测试数据的宽度一样，我们必须确保测试数据和测试目标相关。也就是说，需要有支持特定测试目标的测试数据，相当于针对每种场景都要遍历的测试用例相对应的测试数据。

架构是测试数据的物理结构，它仅与测试目标用于支持测试的任何预先存在的数据相关，例如某个应用程序的数据库或规则表。

测试数据的设计是随着测试用例的设计而产生的，一般从功能测试的测试数据设计和非功能测试的测试数据设计中产生。

3) 测试数据的自动生成

测试数据的自动生成将有效地减轻测试人员的劳动强度，提高测试的效率和质量，节省软件开发的成本。根据估算，对于一个典型的大型软件项目，若能自动生成测试数据，则能节省整个软件开发费用的 4%，这是很可观的。HP 等公司都有对测试数据自动生成的工具支持。

3. 测试脚本

软件测试的最终目的是让用户满意他们所使用的软件，但要使最终用户对软件感到满意，最有力的举措就是对最终用户的期望加以明确阐述，以便对这些期望进行核实并确认其有效性。测试用例反映了要核实的需求。然而，核实这些需求可能通过不同的方式并由不同的测试员来实施。例如，执行软件以便验证它的功能和性能，这项操作可能由某个测试员采用自动测试技术来实现，由测试脚本支撑。

测试脚本一般指的是一个特定测试的一系列指令，这些指令可以被自动化测试工具执行。为了提高测试脚本的可维护性和可复用性，必须在执行测试脚本之前对它们进行构建。或许你会发现这样的情况，即有的操作将出现在几个测试过程中。因此，应有目的地确定这些操作的目标，这样就可以复用它们的实施。

1) 测试脚本的用途

更改目标软件时，需要对测试过程进行局部的可控制的变更。这将使得测试过程和测试脚本对目标软件的变化有更大的应变能力。例如，假设软件的登录部分已经改变。在遍历该登录部分的所有测试用例中，只有关于登录的测试过程和测试脚本需要进行改变。

测试脚本是针对一个测试过程的。一个测试过程往往需要众多的数据来测试。通过自动录制得到的脚本，所有的输入数据都是常数，是固定的。

如果需要使用一个测试脚本测试多组数据，就需要对脚本进行参数化，把固定的常数修改为来自数据源变量。

2) 测试脚本语言

测试脚本语言是脚本语言的一种，准确地讲，是脚本语言在测试领域的一个分支，是自动化软件测试设计的基础。

脚本语言工作的核心是脚本解释器，所有具体指令或函数的执行都由它来完成，扩展项实现了与其他语言的接口，使脚本语言运行 C/C++、Java 等语言中的函数成为可能；同时在用户具体应用中可以定义命令和函数，应用更加灵活；作为解释器，它也提供了基本的内建指令或函数，不同厂商、版本的解释器提供的内建命令(函数)可能不同。

6.2.7　软件测试过程中的配置管理

软件测试需要进行充分的测试准备，需要科学、规范的测试过程管理。有效的配置管理对于跟踪和提高测试质量及效率起到十分重要的作用。测试过程中的配置管理工作不仅包括搭建满足要求的测试环境，还包括获取正确的测试及发布版本。

软件测试配置管理一般应用过程方法和系统方法来建立软件测试管理体系，也就是把软件测试管理作为一个系统，对组成这个系统的各个过程加以识别和管理，以实现设定的系统目标。同时要使这些过程协同作用、互相促进，从而使它们的总体作用大于各个过程之合。软件测试配置管理的主要目标是在设定的条件限制下，尽可能发现和找出软件缺陷。测试配置管理是软件配置管理的子集，作用于测试的各个阶段。其管理对象包括软件测试计划、测试方案(用例)、测试版本、测试工具及环境、测试结果等。

1. 软件测试配置管理的目标和阶段

软件测试配置管理的目标是：①在测试过程中控制和审计测试活动的变更；②在测试过程中随着测试项目的里程碑，同步建立相应的基线；③在测试过程中记录并且跟踪测试过程中的变更请求；④在测试过程中针对相应的软件测试活动或产品，测试人员应将他们标识为被标识和控制并且是可用的。

软件测试配置管理的阶段可划分为：

(1) 需求阶段。我们要进行客户需求调研和软件需求分析。

(2) 设计阶段。在这个阶段我们要进行概要设计和详细设计工作。

(3) 编码阶段。这时我们主要进行的工作是编码。

(4) 测试和试运行阶段。在这个阶段我们要进行单元测试、用户手册编写、集成测试、系统测试、安装培训、试运行和安装运行等工作。

(5) 正式运行及维护阶段。在这个阶段我们对产品进行发布和不断维护。

在软件测试的过程中会产生很多东西，比如测试的相关文档和测试各阶段的工作成果。这些包括测试计划文档、测试用例以及自动化测试执行脚本和测试缺陷数据等。为了以后方便查阅和修改，应该将这些工作成果和文档通过软件测试配置管理保存起来。

2. 测试配置的过程管理

通常测试配置的过程管理包括：

1) 建立配置变更控制委员会

该委员会要做到对项目的每个方面都有所了解。

2) SCM(Software Configuration Management)库的建立和使用

要求在每一个项目过程中都要建立、使用和维护一个软件配置管理库，由配置管理工具支持。这有助于在技术和管理两个方面对所有的配置项进行控制，并且对它们的发布和有效性也能起到控制作用，同时还能对软件配置管理库(SCM 库)进行备份，以防发生意外或风险时，能够作为保存灾难恢复备份的副本。

3) 配置状态报告

配置状态报告是软件测试配置管理过程中的一项重要活动，在软件测试配置管理过程中，配置人员要管理和控制所有提交的产品，然后在有产品提交或变更为完成时，配置人员要进行相应的质量检查。在这之后，配置人员不但要将批准通过的配置项放入基线库中，而且还要记录配置项及其状态，编写配置状态说明和报告。通过配置人员的这些工作来确保所有应该了解情况的组织或个人能够及时地知道相关的信息。

4) 评审、审计和发布过程

为了保持 SCM 库中内容的完整性和质量，要求采取适当的质量保证活动来应对 SCM 库中各项的变化，以此来确保在基线发布之前能够执行审计活动。该活动包括基线审计、基线发布和产品构造。

3. 测试配置管理的主要参与人员及其分工

软件测试配置管理中人员的角色分配和分工是确保配置管理活动在软件开发、测试和维护过程中得到贯彻执行的重要保障，因此在制定测试配置管理计划和开展测试配置管理之前，首先要确定配置管理活动的相关人员以及他们的职责和权限。

1) 项目经理

项目经理作为整个软件的开发及维护活动的负责人，主要职责包括采纳软件测试配置控制委员会的建议，对配置管理的各项活动进行批准，并且在批准之后还要控制它们的进程。项目经理的具体工作职责如下：首先制定项目的组织结构以及配置管理策略；然后要批准和发布配置管理计划；接下来要对项目起始基线和软件开发工作里程碑进行制定；最后要接受并审阅配置控制委员会的报告。

2) 配置变更控制委员会

配置变更控制委员会的职责则是对配置管理的各项具体活动进行指导和控制，并且为项目经理的决策提供建议。该委员会的具体工作职责如下：首先是要批准软件基线的建立以及配置项的标志；然后是制定访问控制策略；接下来是建立、更改基线的设置和审核变更申请；最后是根据配置管理员的报告，从而决定相应的对策。

3) 配置管理员

根据制定的配置管理计划执行各项管理任务，这就是配置管理员的职责，配置管理员要定期向配置变更控制委员会提交报告，同时还要列席他们的例会。配置管理员的具体工作职责如下：①对软件配置管理工具进行日常管理与维护；②提交配置管理计划；③各配置项的管理与维护；④执行版本控制和变更控制方案；⑤完成配置审计并提交报告；⑥对开发人员进行相关的培训；⑦对开发过程中存在的问题加以识别并制定解决方案。

4) 开发人员

开发人员的职责为：在了解了项目组织确定的配置管理计划和相关规定之后，按照配置管理工具的使用模型来完成开发任务。

总之，软件测试配置管理要求做到：①每个测试项目的配置管理责任明确；②配置管理贯穿项目的整个测试活动；③配置管理应用于所有的测试配置项，包括支持工具；④建立配置库和基线库；⑤定期评审基线库内容和测试配置项活动。

4. 测试配置管理中的版本控制

配置管理中的一项重要内容就是版本控制。软件测试的版本控制，简单来说就是对测试有明确的标识、说明，并且测试版本的交付是在项目管理人员的控制之下。用来识别所用版本的状态就是对测试版本的标识、对不同的版本进行编号，对软件质量稳定度趋势的反映也可以由测试的版本控制体现。版本控制是软件测试的一门十分实用的实践性技术，

将各次的测试行为以文件的形式进行记录，并且对每次的测试行为进行编号，标识公布过的每一个测试版本，以此来进行测试排序和管理。

版本控制的作用是跟踪、记录整个测试过程，包括测试本身和相关文档，以便对不同阶段的待测试软件有相关文档进行标识和差别分析，便于协调和管理测试工作。

为了有效地控制软件测试版本，可采用如下方法：

1) 制定合理的版本发布计划，并加强版本控制管理

软件版本的发布应有计划地执行，在项目开发计划中明确发布版本的时机和版本更新的策略，简单来说就是明确在什么条件下可以发布测试初始版本；在什么条件时进行主版本、子版本的更新。这样可以避免频繁地发布测试版本和随意修改版本号，给测试带来混乱。

2) 强化测试准入条件

软件测试的启动是有条件的，在开发人员发布测试版本时，应有相应文档(如自测报告、软件版本说明等)的支持，这是前提，不满足这个前提，测试活动不应启动。

软件版本说明可以使测试人员了解当前版本的具体内容及其与上一版本的差异。

自测报告是对版本质量的保证，避免出现版本质量太差的情况。

3) 强化 bug 管理

充分使用测试管理工具，做好 bug 管理工作。首先，在提交 bug 时，应完整填写 bug 的详细内容，如发现版本、对应人员、发现的模块等；其次，在开发修改 bug 时，应注明修复问题的信息；最后，在测试人员关闭 bug 时，应填写选择关闭 bug 的版本号。这有利于分析不同版本和不同模块的 bug 走势。

4) 做好版本控制的文档管理工作

在每次提交测试版本、执行测试时都会生成相应的文档，以便记录版本信息和测试记录及测试结果等，管理好这些文档，不仅有助于跟踪和监测测试版本的执行，而且也便于测试活动的追溯。

5) 积极解决问题的态度

无论是开发人员还是测试人员，在版本控制过程中都应有积极的态度，遇到问题及时沟通，以高效的方式来解决问题。

对软件测试的版本控制来说，衡量其效果的标准可归根结底为两点：效率和质量。如果版本控制最终使软件测试效率得到提高、使软件质量得到提升，那就是成功的。反之，则是失败的。

6.2.8　软件测试过程中的组织管理

在工程化的软件项目过程中，软件测试活动贯穿整个软件项目的生命周期。独立的软件测试组织始终与设计、实现、维护组织并行工作。软件测试涉及的人力、物力、时间等资源甚至可能超过软件项目总消耗的一半以上。面对极其错综复杂的问题，人的主观认识不可能完全符合客观现实，与工程密切相关的各类人员之间的沟通和配合也不可能完美无缺。因此，在软件生命周期的每个阶段都不可避免地会产生差错。为了尽可能少地产生差

错，尽可能多地找出程序中的错误，生产出高质量的软件产品，加强对测试工作的组织和管理就显得尤为重要。

1. 软件项目测试组织的建立

由于测试的目标是暴露程序中的错误，从心理学角度看，由程序的编写者自己进行测试是不恰当的。因此，软件项目的测试通常由其他人员组成测试团队或建立测试组织来开展测试工作。好处是：①有利于与软件企业项目管理人员、开发人员及用户进行专业交流；②有利于提高对测试工作、测试管理的重要性的认识，以改进我们的测试过程，提高测试的质量；③有利于从理论角度、专业角度来认识软件测试和测试管理。

1) 测试部门的组织形式

好的组织结构，可以更好地发挥人员的能动性，使工作更有效率，也使工作的质量更高。因此，在组建测试团队之初一定要做好规划：首先可根据测试工作范围和目标估计工作量，确定技术要求；然后确定所需的角色和职责，明确岗位、技术和能力需求；最后生成人员配备管理计划。

常见的测试组织结构可分为如下几类：

(1) 烟囱测试组。烟囱测试组分小型和大型两类，测试人员一般由临时人员组成。小型烟囱测试组，通常由两到五人组成，直接向项目经理负责。大型的可以划分为几个小组，设有测试经理。项目经理负责制定测试计划文档。企业没有正规的方法将测试程序、方法、相关的知识经验传递下去，测试质量难以保证。优点是成本低，不需要对测试人员提供培训、生活保障等服务。

(2) 集中测试组。企业成立专职、独立的测试部门，通常由 10 至 30 人组成。集中测试组为每个项目配备几个全职的测试人员，部分企业中可能还负责执行项目中软件质量管理和性能规范制定的工作，可以将相关的知识、经验传递下去。

(3) 独立验证与确认测试组(IV&V 组)。通常由软件开发组织之外的人员或其中的独立人员组成，如转包商。他们参与检查、验证是否遵循标准、进行软件文档的质量保证检测，主要完成系统测试。可以将其看作最苛求的用户。

(4) 系统方法与测试组(SMT 组)。通常作为企业的内部顾问组的方式存在，主要负责方法及标准的知识交流、编制开发和测试指南、开发测试方法、测试工具评估与培训，并与不同的项目组进行协作，对其进行指导。通常不负责具体测试工作的执行，由软件专家组成。

图 6-9 与图 6-10 分别展示了这两种不同的组织形式。

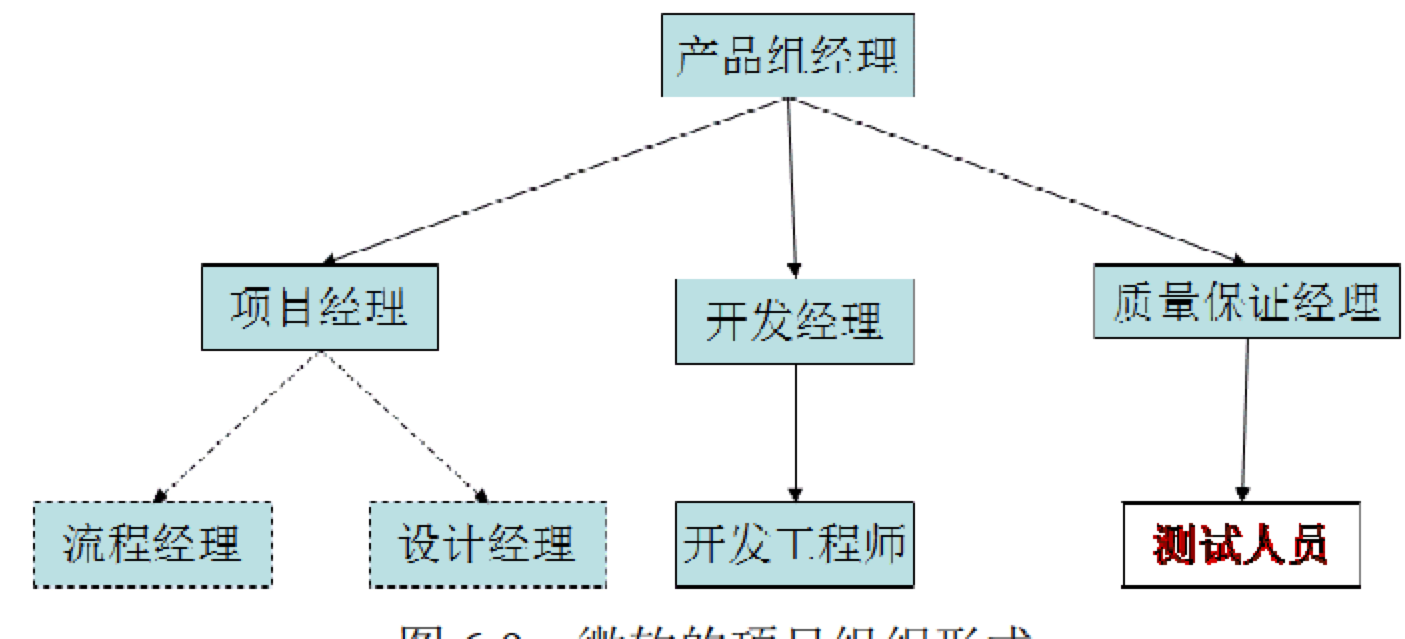


图 6-9　微软的项目组织形式

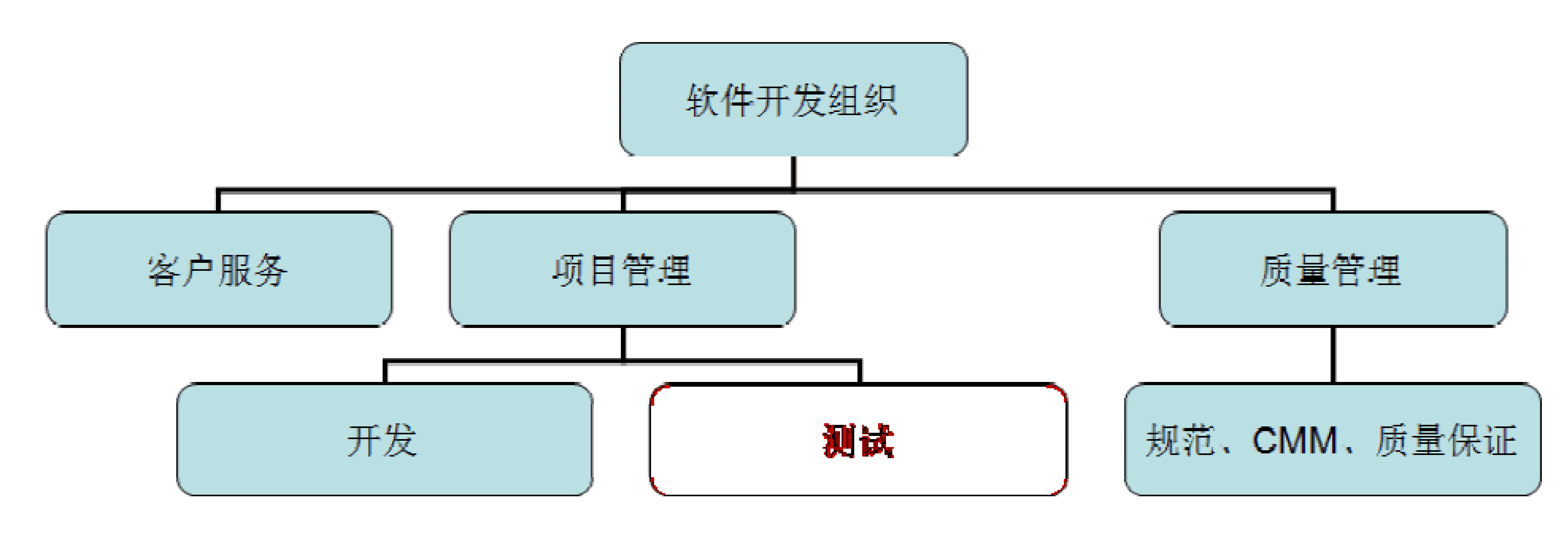


图 6-10　常见的项目组织形式

2) 软件项目测试组织的人员组成

成功的测试组必须在以下 10 个方面满足相关要求：①业务知识(测试工程师应具备业务知识，并和用户紧密接触)；②技术知识(熟悉所测试产品用到的技术，并掌握测试工具、方法等相关技术)；③任务划分(将业务任务和技术任务相互独立)；④资源管理(业务资源和技术资源相互结合)；⑤与开发组的关系(同开发人员协同工作)；⑥生命周期早期介入(测试应在开发周期的早期介入)；⑦测试过程(有成熟的测试过程管理规范)；⑧灵活性/适应性(能够适应不同的测试项目)；⑨度量(掌握度量的方法以改进工作)；⑩过程改进(应致力于工作的不断改进)。

一般情况下，测试组的人员组成包括：①测试经理，负责测试流程、沟通、测试工具的引入、人员管理、测试计划/设计/开发及执行；②测试组长，负责沟通、测试工具引入、人员管理、费用/过程状态报告、测试计划/设计/开发及执行；③测试工程师，负责执行测试计划，进行设计/开发及执行。

测试项目的测试组规模受到很多因素的影响，如企业文化或测试成熟度、测试需求范围、工程师技能水平、测试工具及应用水平、业务知识、组织形式及测试工作介入时间等。

确定测试组规模有如下几种方法：

(1) 开发比例法：根据开发人员数量按照一定比例来确定测试工程师的数量。开发人员指进行设计、开发、编译以及进行单元测试的人员，如表 6-1 所示。

表 6-1　开发比例法

开发类型	开发人员	比例	测试组规模
商业产品(大型市场)	30 人	3:2	20
商业产品(小型市场)	30 人	3:1	10
单个客户端的应用开发	30 人	6:1	5
单个客户端开发并与系统集成	30 人	4:1	7
政府部门应用开发(内部)	30 人	5:1	6
公司应用开发(内部)	30 人	4:1	7

(2) 百分比法：根据测试人员应该占到项目组中人员的百分比数量，如表 6-2 所示。

表 6-2　百分比法

开发类型	项目人员数量	测试组规模比例	测试组规模
商业产品(大型市场)	50 人	27%	13
商业产品(小型市场)	50 人	16%	8
单个客户端的应用开发	50 人	10%	5
单个客户端开发并与系统集成	50 人	14%	7
政府部门应用开发(内部)	50 人	11%	5

(3) 测试程序法：根据测试程序数量以及每个程序可能的执行时间，计算出人小时，再根据完成周期计算测试组规模，如表 6-3 所示。

表 6-3　测试程序法

	测试过程数目	计算因子	人小时	完成周期	测试组规模
历史记录	860	6.16	5300	9 个月	3.4
新项目评估	1120	6.16	6900	12 个月	3.3

(4) 任务计划法：根据历史记录中类似项目的工作量，比较新项目同历史项目的工作量，将历史项目乘以相应的因子。其步骤是先将任务分解，根据历史记录乘以一个因子，计算出新项目的所有任务工作量，然后再根据该工作量和完成周期计算测试组规模。

测试工作对测试组的人员有很多要求，理想状态下应具备：适应各种环境的知识背景，学习速度快，组织能力强，解决问题的能力强，具有创造性，分析和编程能力强，业务领域的知识深，交流与协调能力强，具有测试经验，能够关注细节，文字表达、口头表达能力强等。为此，可采取下列培养途径培养测试人员：

(1) 测试职位：初级测试工程师→测试工程师→高级测试工程师→测试组负责人→测试负责人→测试经理→产品/业务经理。

(2) 技术技能：测试工具→测试自动化编程→编程语言→操作系统→网络、数据库→测试生命周期，需要一至两年。

(3) 测试过程：手工测试，测试用例编写，测试执行，测试文档编写，测试设计，测试计划制定，测试需求分析，评审、制定和改进过程，指导初级工程师工作，了解业务领域，需要三至四年。

(4) 测试组工作：任务安排、跟踪和报告，监管测试工程师，掌握测试周期支持工具，需要四至六年。

(5) 项目管理：管理项目，与客户交流，管理测试人员，需要六至十二年。

(6) 产品管理：项目或产品研发指导，促进产品销售，确定业务机会，承担盈亏责任，需要十二年以上。

2. 测试人员分阶段投入

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对于分析、设计和实现等各阶段得到的结果，包括需求规格说明、设计规格说明及源程序都应进行软件测试。基于此，测试人员的组织也应是分阶段的。

(1) 需求评审：软件的设计和实现都是基于需求分析规格说明进行的。需求分析规格说明是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量，应对其进行严格的审查。审查小组的人员组成有：组长一人，成员包括系统分析人员、软件开发管理者、软件设计/开发/测试人员和用户。

(2) 设计评审：软件设计是将软件需求转换成软件表示的过程，主要描绘出系统结构、详细的处理过程和数据库模式。按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，同时利用关系数据库的规范化理论对数据库模式进行审查。评审小组的人员组成有：组长一人，成员包括系统分析人员、软件设计人员、测试负责人员各一名。

(3) 程序的测试：软件测试是整个软件开发过程中交付用户使用前的最后阶段，是软件质量保证的关键。软件测试在软件生命周期中横跨两个阶段：通常在编写出每一个模块之后，就对它进行必要的测试(称为单元测试)。编码与单元测试属于软件生命周期中的同一阶段。该阶段的测试工作，由编程组内部人员进行交叉测试(避免编程人员测试自己的程序)。这一阶段结束后，进入软件生命周期的测试阶段，对软件系统进行各种综合测试。测试工作由专门的测试组完成。测试组设组长一名，负责整个测试的计划、组织工作。测试组的其他成员由具有一定的分析、设计和编程经验的专业人员组成，人数根据具体情况可多可少，一般三至五人为宜。

3. 测试小组(对应小型烟囱测试组)的运行

测试的工作实体(最小组织单位)是测试小组和支持小组，分别由组长全权负责。组长向测试主管负责。

测试小组内部的角色分为测试人员、测试工程师和测试支持人员(管理人员、测试环境建设人员属于支持人员)。

测试小组根据测试项目或评测项目的需要临时组建，组长也是临时指定。与项目组的最大区别是生命周期短，一般是两周到四个月。在系统测试期间或系统评测期间，测试组长是测试对外(主要是项目组)的唯一接口，对内完全负责组员的工作安排、工作检查和进度管理。

支持小组按照内部相关条例负责测试的后勤保障和日常管理工作，机构设置一般相对比较稳定，主要负责网络管理、数据备份、文档管理、设备管理和维护、员工内部培训、测试理论和技术应用、日常事务管理和检查等。

另外，测试对于每一个重要的产品方向，均设置一至三人来长期研究和跟踪竞争对手的产品特征、性能、优缺点等。在有产品测试时，指导或参加测试(但不一定作为测试组长)，在没有产品测试时，进行产品研究，并负责维护和完善测试设计。目前希望在需求分析阶段多多参与。

测试小组的一般运行方式是：

(1) 项目组提交系统测试申请，给测试部门指定账号(包括版本控制服务器上的位置)，由专人检查文档格式和完备性(文档包括需求文档等相关文档)。

(2) 检查合格后由测试主管审查并通过，成立测试小组，指定测试组长(可以暂时没有组员)。

(3) 测试组长根据该产品的申请报告、测试规范和以往测试数据，制定测试方案、测

试计划。

(4) 测试主管审核通过测试计划和测试方案后，根据测试计划指定测试小组成员，并由支持小组完成其他支持任务(如设备的配备、测试数据库的建立、网络权限的修改)。

(5) 测试期间测试小组根据测试计划进行实际测试，记录并跟踪测试缺陷报告，填写测试记录。测试期间测试组长与项目组(测试经理)经常沟通，并获取产品的更新版本。同时，测试组长审查、修改并提交所有缺陷报告，保证随时掌握产品的质量情况，并监督测试进度。

(6) 产品进行到一定阶段后(标志是测试缺陷报告库中所有的报告处于归档状态)，由项目组和测试组长共同决定产品进入稳定期测试。稳定期测试版本之前的版本必须在显著位置标明测试版字样。

(7) 稳定期测试期间所发现的缺陷报告也需要记录在测试缺陷报告库中，并在稳定期结束后由双方(有时可能也有市场方面的意见)共同决定对这些缺陷的处理方式。如果需要改动产品，就重新开始稳定期，否则通过稳定期测试。

(8) 测试组长对于通过稳定期测试的产品填写综合测试报告，测试中心依此发布产品发行通知。

(9) 测试小组对整个测试过程和产品质量进行总结和评价，形成文档并备案。同时，将测试过程中对测试设计的改动纳入基线。最后，组长整理并在指定地点保存相关测试用例、测试数据以及测试文档。

(10) 测试部门解散测试小组。在解散测试小组以前一定进行工作总结(每个人都要总结，总结测试过程中使用新的方法、技巧或某些方法、技巧的心得，以及测试过程中的失误)，之后，将其形成文档或程序归档。

另外，在系统测试阶段，也将要求测试小组进行一些常规内容测试(如早期的 Y2K 测试、病毒检查、裸机测试、加密检查、说明书检查等)，并要求写入测试方案。

6.3　软件测试管理工具

对于测试管理来说，有许多令人生畏且不可避免的挑战，好消息是有大量管理工具可以应对这些挑战。使用测试管理工具对软件的整个测试输入、执行过程和测试结果进行管理，可以提高测试的效率、测试时间、测试质量、用例复用、需求覆盖等。

由于软件项目测试越来越复杂，单纯增加测试人手已不能解决问题，需要软件工具来管理测试。软件测试工具种类繁多，主流的测试工具可以分为测试管理工具、负载测试工具、功能测试工具等。相比于测试管理工具，其他具体的测试技术和工具都是次要因素，比如各种测试工具、白盒测试、黑盒测试等。这些具体的测试技术都是相对比较固化的，测试技术人员可以慢慢地掌握它们。但在评价一支测试团队的测试水平，或者在评估一个项目的测试质量时，不会因为这些具体的测试技术掌握与否而断言，更多考虑应是测试管理过程和软件测试管理工具的运用。

6.3.1　软件测试管理工具应具备的功能

一个完整的软件测试管理工具，应能用于测试的计划、文档和缺陷跟踪等各种测试行为的管理，并能提供对人工测试和自动测试基于过程的分析、设计和管理功能，把应用程序测试中涉及的全部任务集成起来；包括测试中包含的所有工作，跟踪测试资产中的依赖关系和相互关联，并能对质量目标进行定义、测量和跟踪。

一般而言，软件测试管理工具应具备如下功能：

1. 测试计划和进度管理

软件测试管理工具应能跟踪项目测试工作是否按照预期计划和进度开展，对软件产品从功能分类、相关人员分配、测试用例编写、测试用例执行到问题报告生成的整个测试流程进行系统、有效的管理。项目管理人员可以通过该工具随时了解、监控被测项目的执行进度和软件问题的处理状态，为测试人员和开发人员的工作提供有效的考核依据，保障软件开发和测试的顺利进行。

2. 测试资产管理

包括测试计划、测试用例、测试脚本、测试报告的创建与维护以及缺陷跟踪，保证测试资产之间是可跟踪的、一致的。无论是开发人员、测试人员还是项目管理人员，都可以随时编写、修改和查阅测试用例和软件问题报告，能对测试用例和问题报告进行长期保存以避免流失。对测试用例的编写与执行情况进行全程记录，便于追踪测试用例在各个测试阶段的执行过程，及时调整测试策略与方法。最后，还应提供统一的软件问题报告模板与测试用例模板，使测试人员能够更加准确、详细地编写测试用例和描述软件问题，保证测试用例与软件问题报告描述的一致性，便于积累、分类与查询。

3. 测试质量评估与报表

软件测试管理工具应具有实用的统计功能，可以从各种角度建立分析统计报表，以便及时掌握测试执行情况，例如，软件问题的有效发现率、有效修复率及各项测试工作的进度。好的软件测试管理工具必须提供所有相关信息的完整且正确的报告，所有这些测试报告要指导如何对测试工作的不同结果进行分析和沟通，并为不同的项目角色能随项目的进展对变化做出合适反应提供决策依据，以及帮助判断测试的当前状态和质量水平。

6.3.2　软件测试管理工具的选择

面对市场上众多的商业化软件测试管理工具，选择就成了一个比较重要的问题。在选用软件测试管理工具的时候，建议从以下几个方面来权衡和考虑。

1. 功能考量

功能当然是人们最关注的内容，选择一个测试工具首先就是看它提供的功能。当然，这并不是说测试工具提供的功能越多就越好。在实际的选择过程中，适用才是根本。钱要花在刀刃上，为不需要的功能花费金钱实在不是明智的行为。事实上，目前市面上同类软件测试工具之间的基本功能都是大同小异，各种软件所提供的功能也大致相同，只不过侧重点不同。例如报表功能，查看测试管理工具所生成报告的人员不一定对测试很熟悉；因

此，测试工具能否生成合乎要求的结果报表，能够以什么形式提供报表是其中一个考查因素。

2. 测试管理工具的集成能力

测试管理工具的集成能力也是必须考虑的因素，这里的集成包括两方面的意思：首先，测试管理工具能否和开发工具进行良好的集成；其次，测试管理工具能否和其他测试工具进行良好的集成。另外，与操作系统和开发工具的兼容性也需要考虑。测试管理工具可否跨平台，是否适用于公司目前使用的开发工具，这些也都是在选择一个测试管理工具时必须考虑的问题。

3. 能否与现有的测试管理保持连续性和一致性

引入测试管理工具的目的是使测试管理自动化，引入工具时需要考虑工具引入的连续性和一致性。也就是说，对测试工具的选择必须要有全盘的考虑，要考虑到公司的实际情况，避免盲目引入测试管理工具。因为并不是每种测试工具都适合公司目前的实际情况，如果怀着美好的愿望花了不小的代价引入测试工具，一年半载后测试工具却成了摆设，就不仅浪费了资金，而且还会对软件项目的开发进度产生严重影响。

4. 是否具备测试团队管理功能

测试管理工具应具有测试团队管理功能，要能根据人员的分工和职能不同来严格划分权限，从而明确测试任务，并保证测试质量，如测试人员的绩效考核、人员的技术定位、人员激励等。

测试管理工具应能有效处理人力资源不足问题或能更充分地利用人员，能协调运用任何人力资源，而不管它们在什么地方。这些重要的人力资源可能存在于不同的地方，这需要仔细有效的协同合作，才能使大多数测试人员和其他人员参与到测试管理中，对于那些跨越一个或更多个场所或测试团队的项目来说，还包括区域场所和团队协调。

6.3.3　常用软件测试管理工具介绍

在软件测试管理过程中，免不了要借助一些软件测试管理工具，以对测试进行管理。一般而言，软件测试管理工具用于对测试计划、测试用例和测试实施进行管理，另外还包括对缺陷的跟踪管理。具有代表性的商用软件测试管理工具有 IBM Rational 公司的 Test -Manager、Compureware 公司的 TrackRecord、HP 公司的 ALM 等，深受中小企业欢迎的开源软件测试管理工具有 TestRunner(更名为 Testopia)和 TestLink，国产的软件测试管理工具有上海泽众软件科技有限公司的TestCenter。

1. IBM TestManager

IBM TestManager 的优点是有自己的客户端、响应速度较快、Case 管理功能强大(可以任意地组合自己的 Test Case；可以区分不同 build 保存的测试结果、集成了强大的数据生成器(datapool)、与第三方报表完全集成、可生成多种且完整的数据报表及图表、可与 IBM 的其他组件(如 CQ、Robot、Requisite Pro 等)无缝结合，总之是很完整的测试管理及测试执行平台。

其不足之处是：没有权限管理功能；支持的数据库类型太少，尤其是不支持 Access 数据库，不能支持太多的并发访问用户数；不支持 Test Case 的实例化功能；不能预先安排 Case 的执行任务给相应的测试人员；有用户反映有莫名其妙的错误，找不到解决方案。

2. HP ALM

HP ALM 的优点是有些功能比 IBM TestManager 更强大，如可以进行需求管理和缺陷管理；有测试用例执行跟踪的功能，可以统计测试用例对需求及缺陷的覆盖；可以和 VSS 集合，对测试用例进行版本控制；有灵活的缺陷定制，可以和 IBM ClearQuest 紧密结合，使两个缺陷工具的数据得到同步；采用 Web 形式的界面，界面较友好。

其不足之处是：每个项目库对同时在线的人数有限制；可能存在部分不稳定性，但是基本功能没有问题；对 Close 的测试集没有状态标记；快速执行测试用例的时候，不能看到测试用例的内容，而快速执行是经常用到的功能；不能实现自动多级连动，需要硬编码。

3. TestRunner

这是与 Bugzilla 集成的一款测试管理工具，使用时有一定的局限性，但其易用性的功能，特别是与 Bugzilla 共同进行的管理效果显著，因而受到许多测试工程师的青睐。其优点是：开源免费；采用 Web 方式的管理界面；自动邮件提醒；和缺陷管理系统 Bugzilla 结合紧密，有测试用例执行管理；测试用例可以分优先级；测试用例可以有评审功能(测试用例有不同的状态)。

其不足之处是：安装设置较烦琐，测试用例必须按照一个步骤对应一个验证点的形式来编写。

4. TestLink

这是一款优秀的开源测试管理软件。与 TestRunner 不同，TestLink 可以和更多的缺陷管理软件进行集成(BugZilla、Mantis 等)。因此，如果选用了其他的缺陷管理软件，则推荐使用 TestLink 进行测试。TestLink 用于测试过程中的管理，通过使用 TestLink 提供的功能，可以将测试过程从测试需求、测试设计到测试执行的完整流程管理起来。同时，它还提供了测试结果的多种统计和分析图表，使开始测试工作和测试结果分析工作变得更简单。TestLink 是 SourceForge 的开源项目之一。

5. TestCenter

TestCenter是由上海泽众软件科技有限公司研发的一款功能强大的测试管理工具，可以帮助用户实现测试用例的过程管理，对测试需求过程、测试用例设计过程、业务组件设计实现过程等整个测试过程进行管理。通过实现测试用例的标准化(即每个测试人员都能够理解并使用标准化后的测试用例)，降低了测试用例对个人的依赖；提供测试用例复用，使测试用例和测试脚本能够被复用，以保护测试人员的资产；提供可伸缩的测试执行框架，提供自动测试支持；提供测试数据管理，帮助用户统一管理测试数据，降低测试数据和测试脚本之间的耦合度。

(1) 测试需求管理：支持测试需求树，树的每个节点是一个具体的需求，也可以定义子节点作为子需求。每个需求节点都可以对应到一个或多个测试用例。

(2) 测试用例管理：测试用例允许建立测试主题，通过测试主题来设置测试用例的使用范围，实现有效的测试。

(3) 测试业务组件管理：支持测试用例与业务组件之间的关系管理，通过测试业务组件和数据搭建测试用例，实现了测试用例的高度可配置和可维护性。

(4) 测试计划管理：支持测试计划管理、测试计划多次执行(执行历史查看)，测试需求范围定义、测试集定义。

(5) 测试执行：支持测试自动执行(通过调用测试工具)；支持在测试出错的情况下执行错误处理脚本，保证出错后的测试用例脚本能够继续被执行。

(6) 测试结果日志查看：提供截取屏幕的日志查看功能。

(7) 测试结果分析：支持多种统计图表，如需求覆盖率图、测试用例完成的比例分析图、业务组件覆盖比例图等。

(8) 缺陷管理：支持从测试错误到曲线的自动添加与手工添加；支持自定义错误状态、自定义工作流的缺陷管理过程。

6.3.4　应用软件测试管理工具 TestLink

作为基于 Web 的测试管理系统，TestLink 的主要功能包括：测试需求管理、测试用例管理、测试用例对测试需求的覆盖管理、测试计划的制定、测试用例的执行以及大量测试数据的度量和统计。TestLink 能够加速并简化测试的这个管理流程，动态地收集、组织测试用例，跟踪与整合相关联的测试，获取并报告详细的信息，帮助开发人员管理这个测试流程，并且还能够帮助用户自定义以适用项目需求与过程。

1. TestLink 功能介绍

TestLink 针对整个测试过程进行管理，包括创建测试脚本、执行测试、跟踪测试结果等。除此之外，它还能够让测试、测试开发和测试报告变得更加简单。其功能主要表现在：动态地收集和组织测试用例，跟踪测试执行后的测试结果，跟踪独立测试的准确信息，获取并详细地报告测试结果，可以帮助测试人员更好地管理整个测试过程等。

TestLink 在以上功能的表现上自然有它自身的特点，主要表现在：Web 方式访问；测试计划中每个产品的测试都遵循测试流程；用户可以自定义角色；关键字被用于支持深层次的测试组织；测试可以根据优先级指派给测试员，定义里程碑；提供测试报告，支持将文档以 HTML、Word 或 Excel 格式导出，可以直接通过这个工具将测试报告以邮件形式发送出去；支持本地化和国际化；可结合通用的 bug 跟踪系统，如 Bugzilla、Mantis 和 Jira；基于测试的需求管理。

2. TestLink 应用环境的建立

TestLink 是一个开源软件，采用 PHP 开发。因此，如果想使用 TestLink，则至少需要安装 PHP，否则无法使用。另外，TestLink 是一个测试管理平台，是一个 B/S 模式的系统，因此在搭建 TestLink 的过程中，还需要有一台服务器来放置 TestLink 的服务。可以选用的有 IIS，也可以使用 Apache HTTP 服务器。最后，由于测试管理和数据息息相关，需要选用一个数据库来存放测试数据、用户信息、项目信息等，在这里选用 MySQL 数据库。

关于 Apache+MySQL+PHP 的安装方法，在前面介绍安装 Mantis 工具的时候就已介绍过，现在不再详细说明。

下面介绍 TestLink 的安装。

1) 解压TestLink源文件

从 http://sourceforge.net/projects/testlink 上下载 Testlink 的最新版本，目前的最新版本是 1.9.14，将其解压缩，重命名为 testlink，复制到 E:\xampp\htdocs 目录下。

2) 安装TestLink

完成上述步骤后，打开浏览器，在地址栏中输入 http://localhost/testlink 后回车，进入安装页面，如图 6-11 所示。

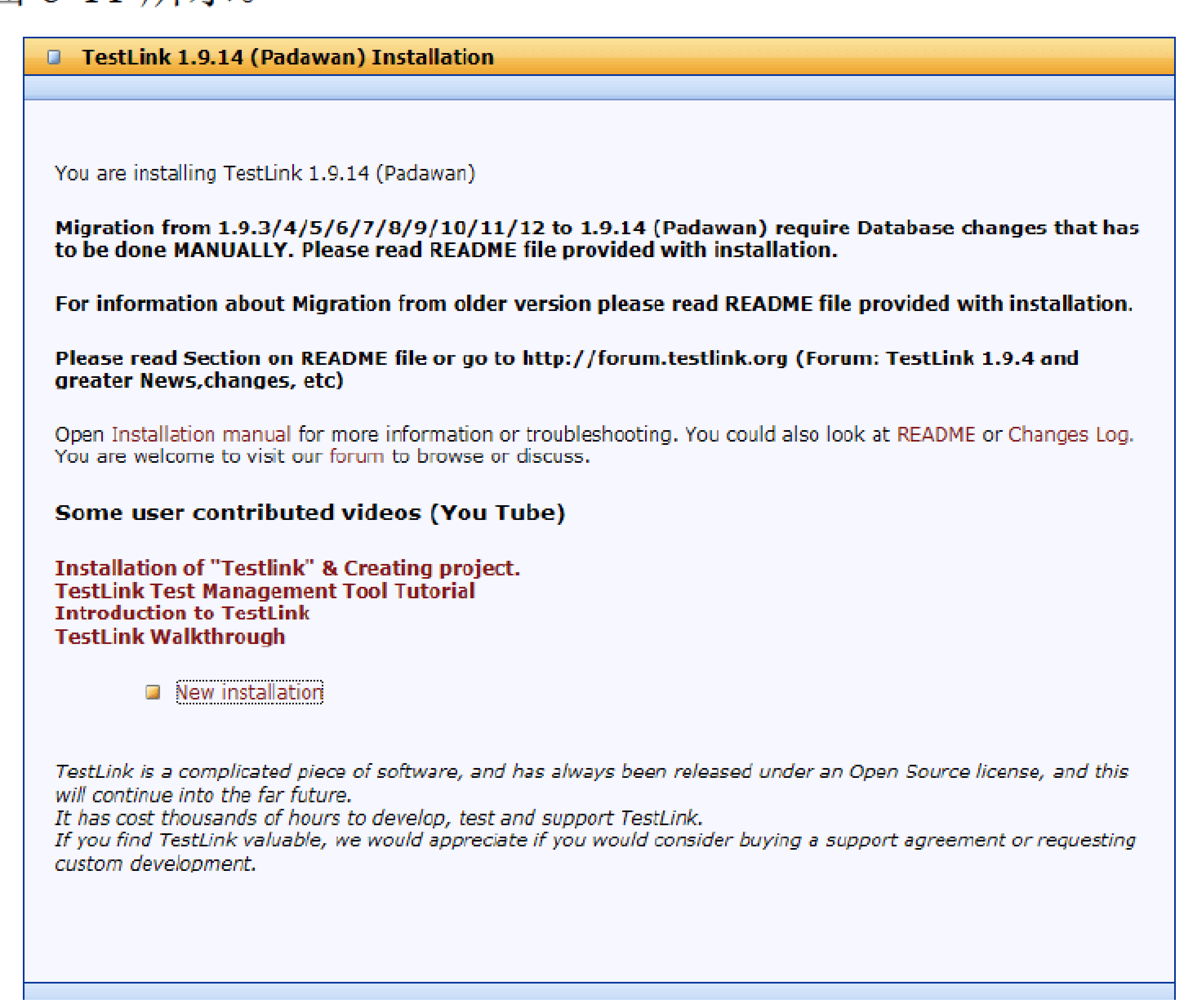


图 6-11　TestLink 安装页面

选中 New installation 复选框，进入安装页面。TestLink 在检查系统配置时，会出现报错(这个问题只会出现在 1.9.4 及之后的版本中)，如图 6-12 所示。

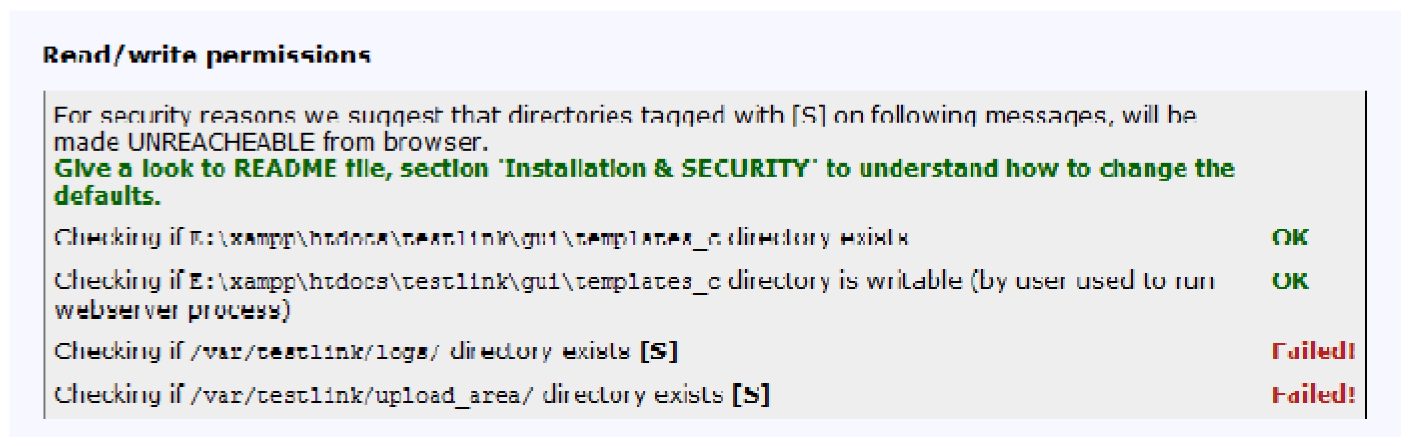


图 6-12　系统检测报错

解决方法：

(1) 打开 testlink 目录下的 config.inc.php 文件，找到以下语句：

```
$tlCfg->log_path = '/var/testlink/logs/'; /* unix example */
```

注释掉该语句，添加如下内容：

```
$tlCfg->log_path = 'testlinkDir/logs/';
```

(2) 找到如下语句：

```
$g_repositoryPath = '/var/testlink/upload_area/';    /* unix example*/
```

注释掉该句，添加如下内容：

```
$g_repositoryPath = 'testlinkDir/upload_area/';
```

注意：

testlinkDir 表示安装路径，本次安装为 E:/xampp/htdocs/testlink。

做出上述修改后，当再次检测时，成功通过，如图 6-13 所示。

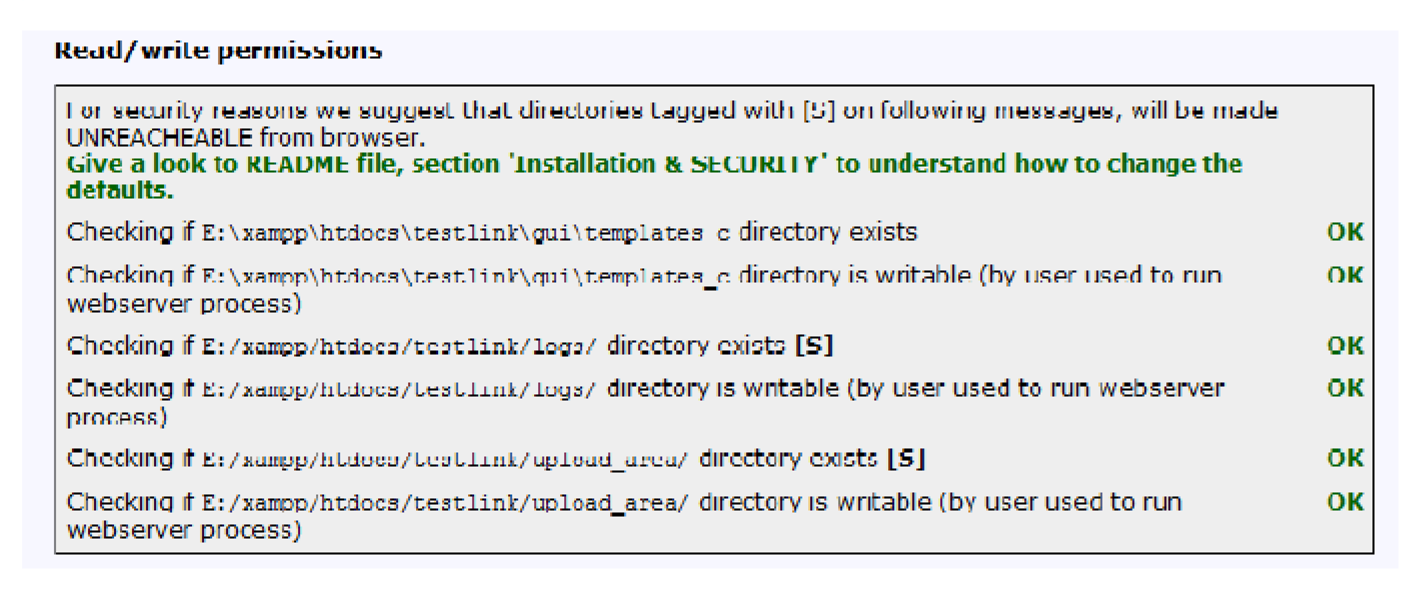


图 6-13　系统检测通过

单击 continue，进入数据库配置页面，安装页面上原有的数据可以保持默认，在其余的填写项中输入数据库的用户名和密码(也就是 MySQL 数据库的用户名和密码)，如图 6-14 所示。单击“下一步”按钮即可开始安装 TestLink。

Table prefix　(optional)

Note: This parameter should be empty for the most of cases.
Using a Database shared with other applications: Testlink can be installed (using this installer) on a existing database used by another application, using a table prefix.
Warning! PART OF INSTALLATION PROCESS CONSISTS on dropping all TestLink tables present on the database/schema (if any TestLink table exists). Backup your Database Before installing and load after this process.

Set an existing database user with administrative rights (root):

Database admin login　root
Database admin password　••••

This user requires permission to create databases and users on the Database Server.
These values are used only for this installation procedures, and is not saved.

Define database User for Testlink access:

TestLink DB login　root
TestLink DB password　••••

This user will have permission only to work on TestLink database and will be stored in TestLink configuration.
All TestLink requests to the Database will be done with this user.

After successfull installation You will have the following login for TestLink Administrator:
login name: admin
password : admin

图 6-14　数据库文件配置页面

最后，转入安装成功页面，如图 6-15 所示。

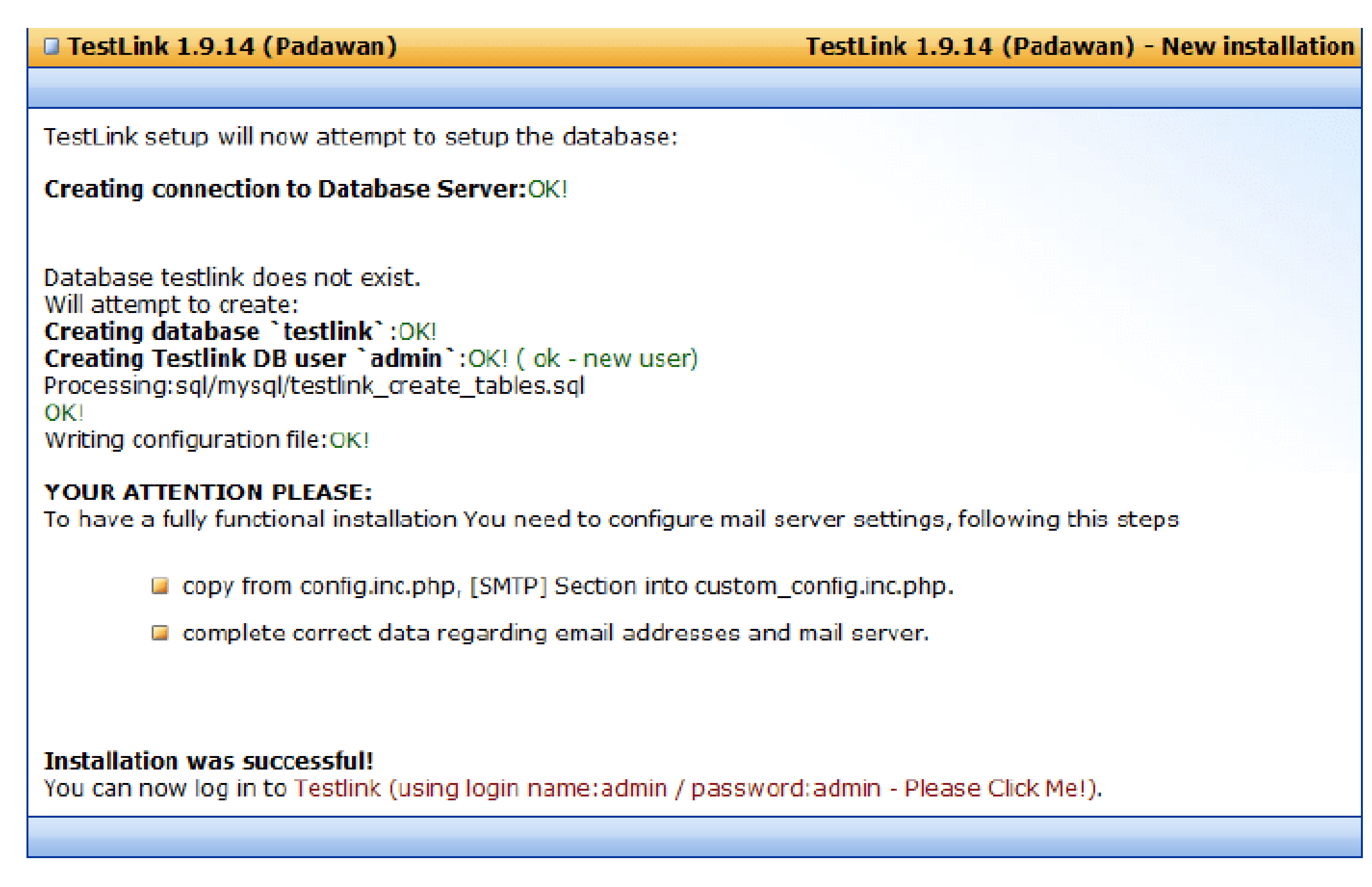


图 6-15　安装成功页面

3) 查看安装是否成功

访问 http://localhost/testlink/index.php，如果能显示如图 6-16 所示的登录界面，那么就说明安装成功了。



图 6-16　启动 TestLink

3. TestLink 使用流程

基于 TestLink 的测试管理流程一般包括：创建项目、创建测试需求、创建测试计划、创建测试用例、为需求添加用例、为计划添加用例、分配测试任务、执行测试任务/报告 bug 并跟踪、查看分析结果，详细流程如图 6-17 和图 6-18 所示。

其中，在创建测试用例之前，需要确定测试用例所隶属的 test project 和 test suite 都已经存在。因为 TestLink 用例数的层次为 test project-test suite-test case，而 test project 在一开始就已经创建，所以现在需要判定 test suite 是否已经创建。

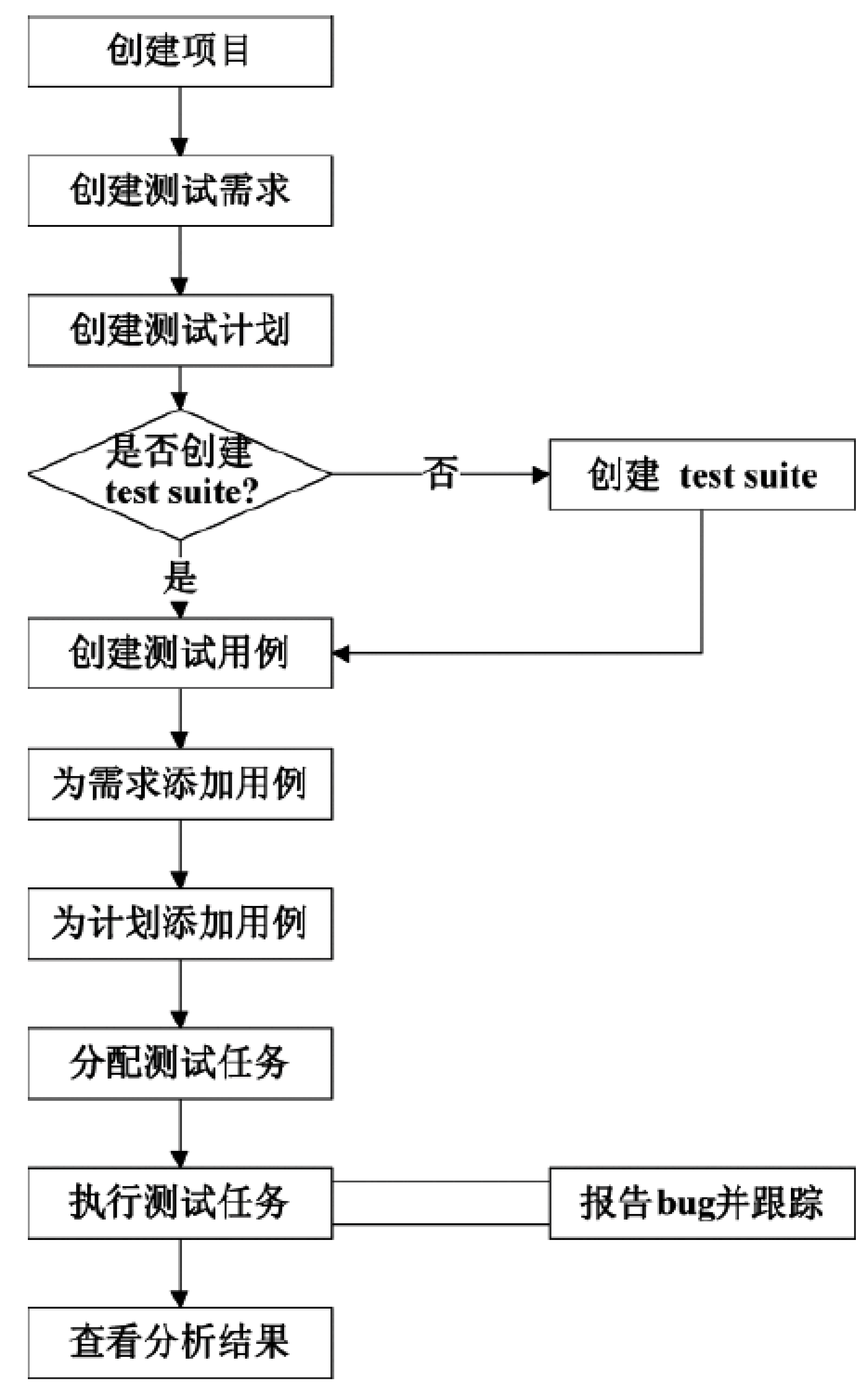


图 6-17　基于 TestLink 的测试管理流程

6.3.5　TestLink 应用举例

下面将从 TestLink 初始配置、确定测试需求、测试用例管理、制定测试计划、测试报告与度量，以及测试结果分析等方面对 TestLink 的具体应用进行介绍，TestLink 的工作流程如图 6-18 所示。

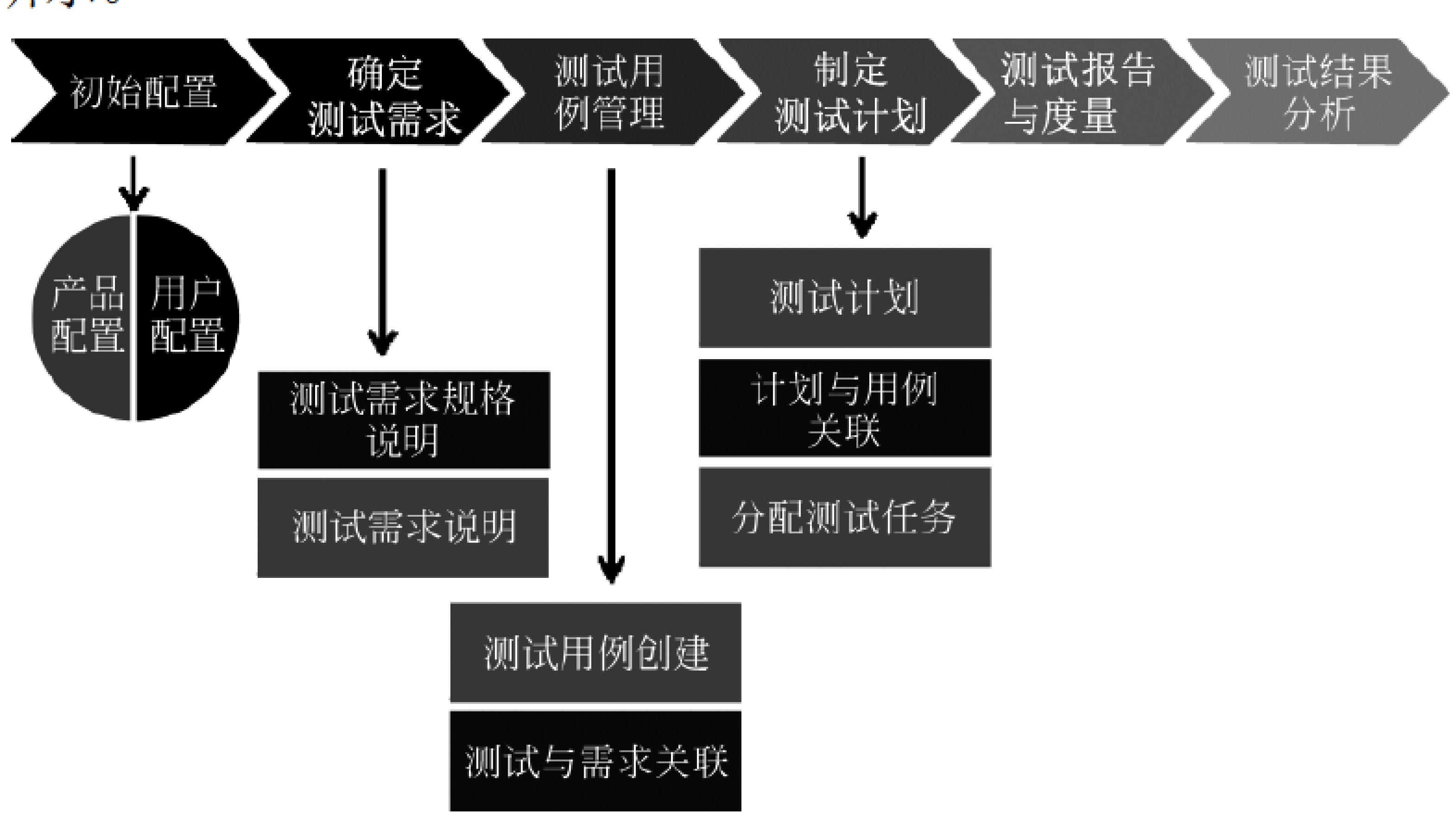


图 6-18　TestLink 的工作流程详解图

1. 初始配置(设置邮箱、用户、产品)

1) 邮箱配置

打开 E:\xampp\htdocs\testlink 目录下的 config_inc.php 文件，找到以下语句并进行修改(以 163 邮箱为例)：

```
$g_smtp_host        = 'smtp.163.com';          #SMTP 服务器
$g_tl_admin_email    = 'xxx@163.com';          #管理员邮箱
$g_from_email        = 'xxx@163.com';          #发送邮箱
$g_return_path_email  = 'xxx@163.com';
$g_mail_priority = 5;                          #邮件级别设定：1 为紧急、5 为不紧急、0 为空
$g_phpMailer_method =PHPMAILER_METHOD_SMTP;
$g_smtp_username     = 'xxx';                  #只在 SMTP 服务器需要用户名密码验证时填写
$g_smtp_password     = 'yyy';                  #只在 SMTP 服务器需要用户名密码验证时填写
$g_smtp_connection_mode = '';                  #默认为空，可以设置 SSL、TLS
$g_smtp_port = 25;
```

2) 用户配置

打开浏览器，在地址栏中输入 http://localhost/testlink。

系统为 TestLink 创建了一个默认的管理员账号，用户名/密码为 admin/admin。可以使用这个账号访问 TestLink，如果是第一次访问，访问后 TestLink 会要求用户创建一个新的测试项目，如图 6-19 所示。

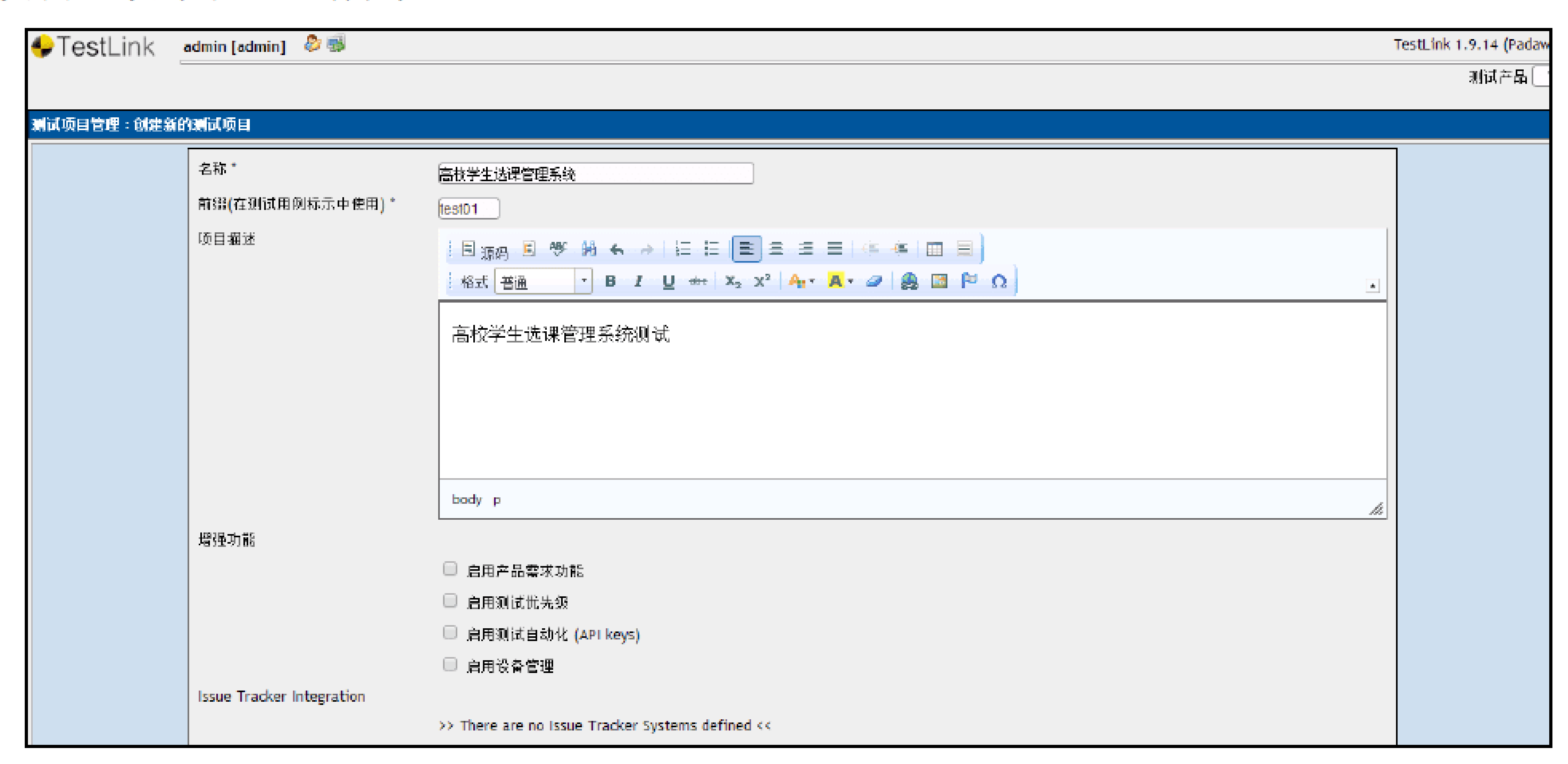


图 6-19　创建新的测试项目

只有在创建了测试项目之后，页面上才会出现功能栏，如图 6-20 所示。

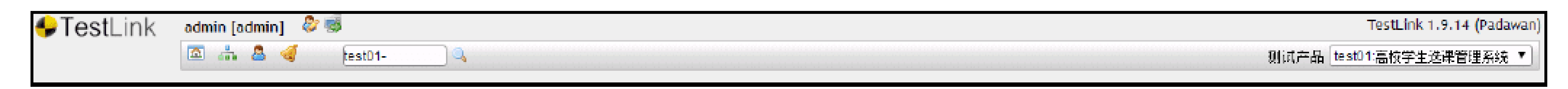


图 6-20　TestLink 的功能栏

在 TestLink 系统中，每个用户都可以维护自己的私有信息。admin 可以创建用户，但无法看到其他用户的密码。在用户信息中，需要提供 e-mail 地址，如图 6-21 所示。这样当用户忘记密码时，便可以通过 e-mail 来获得。

TestLink 有 6 种不同的默认权限级别，对于管理员来说，通过用户管理链接可以很容易地改变权限。这些权限如下所示：

(1) Guest：只能查看测试用例和项目度量。

(2) Tester：只能执行分配给他们的测试用例。

(3) Test Designer：可以开展测试用例和测试需求的所有工作。

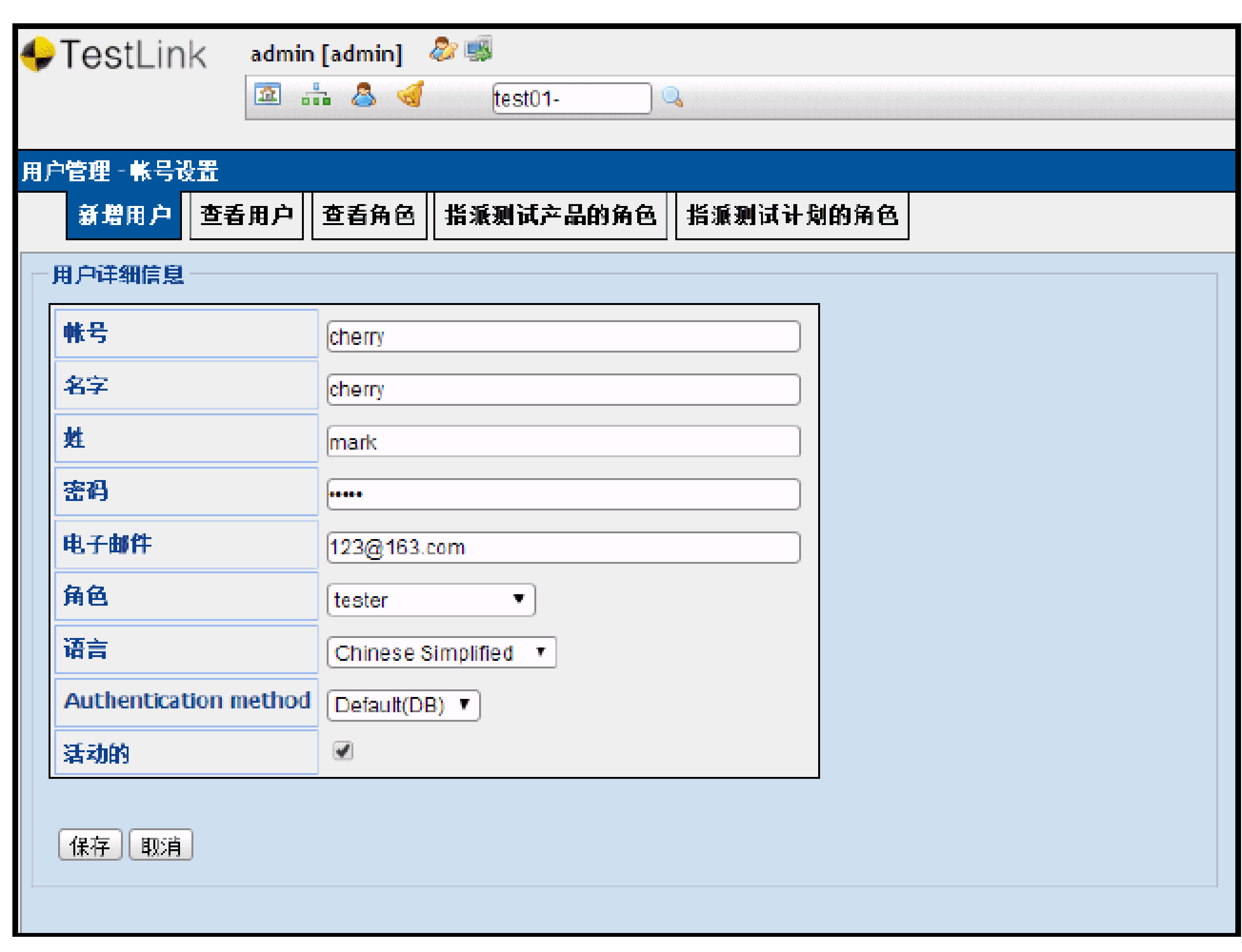


图 6-21　用户信息

(4) Senior Tester：可以查看、创建、编辑和删除测试用例，并且可以执行测试用例，但是不能管理测试计划、管理产品、创建里程碑或分配权限(针对初级测试员和高级测试员)。

(5) Leader：拥有 Tester 的所有权限，并且可以管理测试计划、分配权限、创建里程碑和管理关键字。

(6) Admin(Administrator)：拥有 Leader 的所有权限，并且可以维护整个产品。

详细情况参见表 6-4 和图 6-22。

表 6-4　用户角色及相应职能

角　色	权 限 列 表	能　力
Guest	查看测试规范(组件、分类和测试用例的数据)，查看关键字，查看度量	只能浏览数据
Test Executor Tester	执行测试用例，查看度量	只能执行测试
Test Analyst Senior Tester	执行测试用例，查看度量，创建构建，查看、修改测试规范(组件、分类和测试用例的数据)，查看关键字，创建、编辑、结合和删除测试需求，查看测试需求	编辑测试规范和执行需求
Test Designer	查看度量，查看、修改测试规范(组件、分类和测试用例的数据)，查看关键字，创建、编辑、结合和删除测试需求，查看测试需求	编辑测试规范和测试需求
Test Leader	执行测试用例、创建构建、查看度量，创建、编辑、删除测试计划，设置风险/所有权，编辑/删除里程碑，编辑用例集，设置查看项目的权限，查看/修改关键字，查看/修改测试需求	拥有所有测试计划权限，编辑测试规范和执行测试
Administrator	执行测试用例、创建构建、查看度量，创建、编辑、删除测试计划，设置风险/所有权，编辑/删除里程碑，编辑用例集，设置查看项目的权限，查看/修改测试规范(组件、分类和测试用例的数据)，查看/修改关键字，查看/修改测试需求，创建、编辑和删除产品，创建、删除和维护用户	进行一切可行性操作，只有此用户可以维护产品和用户

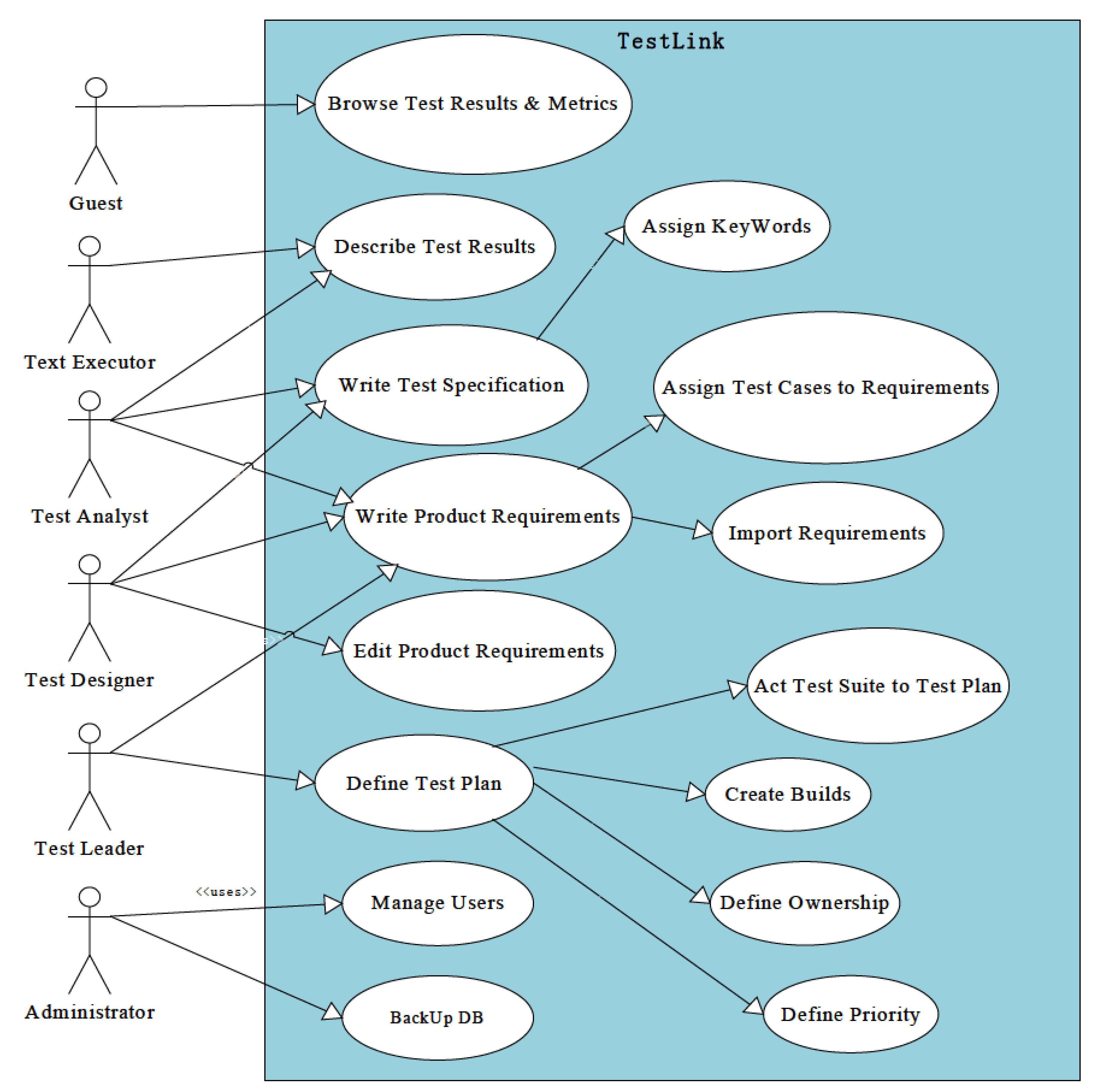


图 6-22　用户角色分类

同时，TestLink 初始配置支持不同地域用户对不同语言的需求，为用户提供不同的语言支持。

3) 产品配置

创建新产品需要有管理员权限，而且不能创建相同名称的产品，但为了使其显示更清晰，可以为产品分配背景颜色，如图 6-23 所示。

创建新产品时要注意以下几点：

(1) 从系统中删除产品本身是不被认可的，因为产品的删除会使很多测试用例处于孤立的状态或者导致这些测试用例也被删除。

(2) 测试计划在特定时间里描绘产品的测试。这句话的意思就是说，所有的测试计划需要根据产品测试用例来创建。

(3) TestLink 可以把数据输入到一个产品中，数据以 CSV 形式读入并在输入环节被进一步说明。

(4) TestLink 可以对多个产品进行管理，Admin 对产品进行设置后，测试人员就可以进行测试需求、测试用例、测试计划等相关管理工作了。TestLink 支持对每个产品设置不同的背景颜色，以便于产品的管理。

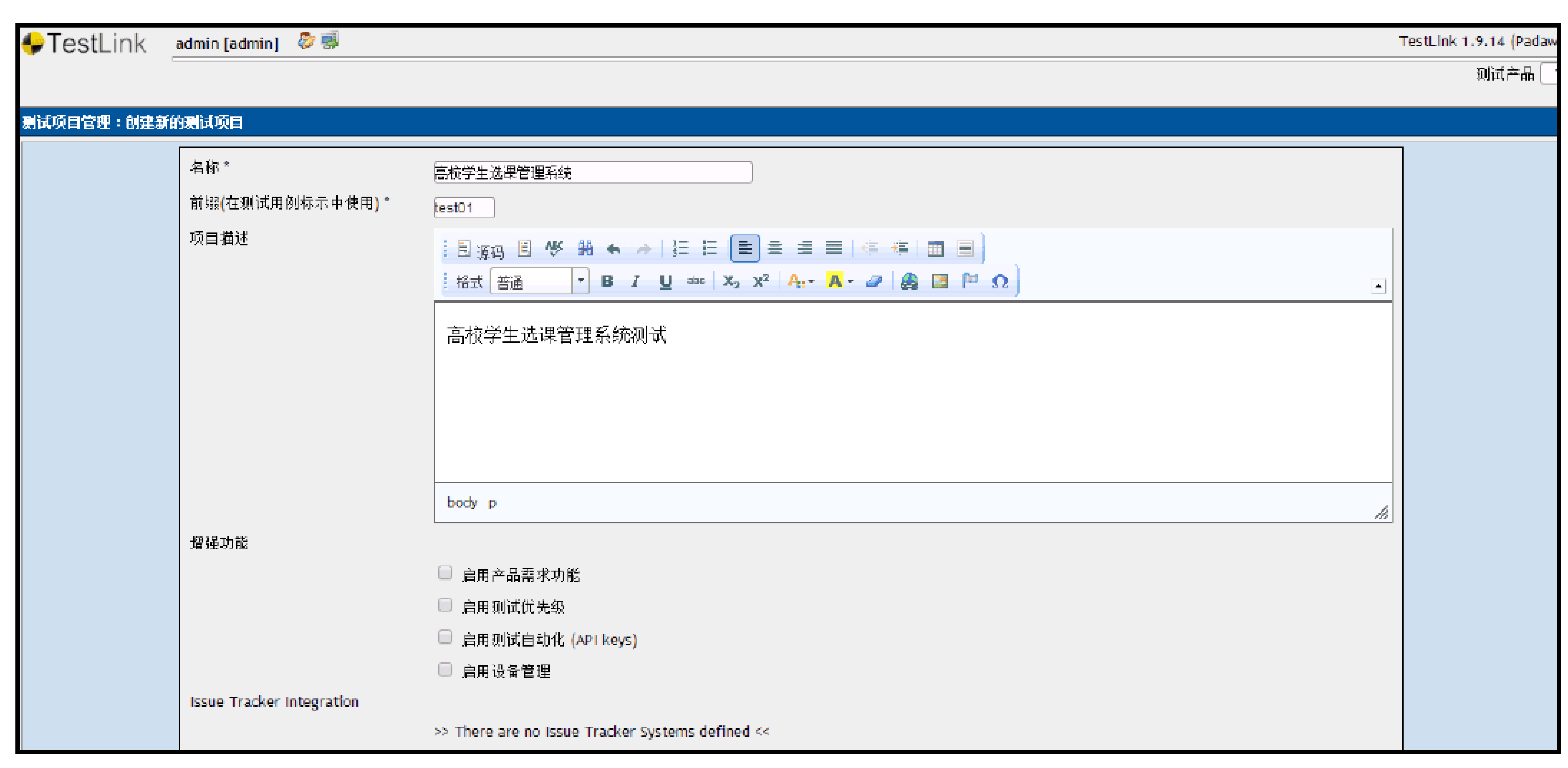


图 6-23　产品管理

2. 确定测试需求

为了验证一个系统是按照指定要求建立的，测试人员需要将测试与需求对应起来。对于每一个需求，可以设计一个或更多个测试用例。在测试执行的最后阶段，测试经理汇报测试的执行情况以及需求的覆盖率。基于这些汇报信息，客户和投资人决定系统是继续测试还是投入使用。为了保证系统是按照指定要求建立的，测试经理必须将风险和测试需求结合起来统筹考虑，以保证系统是按照客户和投资人的指定要求建立的。这样做有下列好处：

(1) 风险和需求相结合可以揭露潜在的或遗漏的需求，这对高风险系统来说尤为有意义。

(2) 测试可以首先集中到信息系统中最重要的部分，首先涵盖最高优先权的风险。

(3) 促使测试人员按照客户和投资人的眼光来看待问题。这使得报告测试项目的状况更加容易。除此之外，还将有充足的依据决定是否进行更多测试还是冒点儿风险。

(4) 风险及其优先权使测试项目在面对压力时的谈判更加容易。例如，在这个测试项目之内，什么风险必须覆盖，哪些可以被延期。将风险与需求相结合进行测试能更加有效地控制测试项目，还能改善客户与股东之间的沟通。测试经理首先测试最高优先权的风险，测试过程将更有效，结果也更可靠。

1) 创建需求规格

单击主页，在主页上找到产品需求，新建需求规格。对需求规格的描述比较简单，内容包含文档 ID、标题、范围，如图 6-24 所示。

单击左侧的“高校学生选课管理系统”节点。在右侧的界面中单击“新建产品需求规格按钮。输入文档 ID“1”，标题“专业管理功能”。然后创建另一个规格文档 ID“2”，标题“课程管理功能”，完成以后如图 6-25 所示。

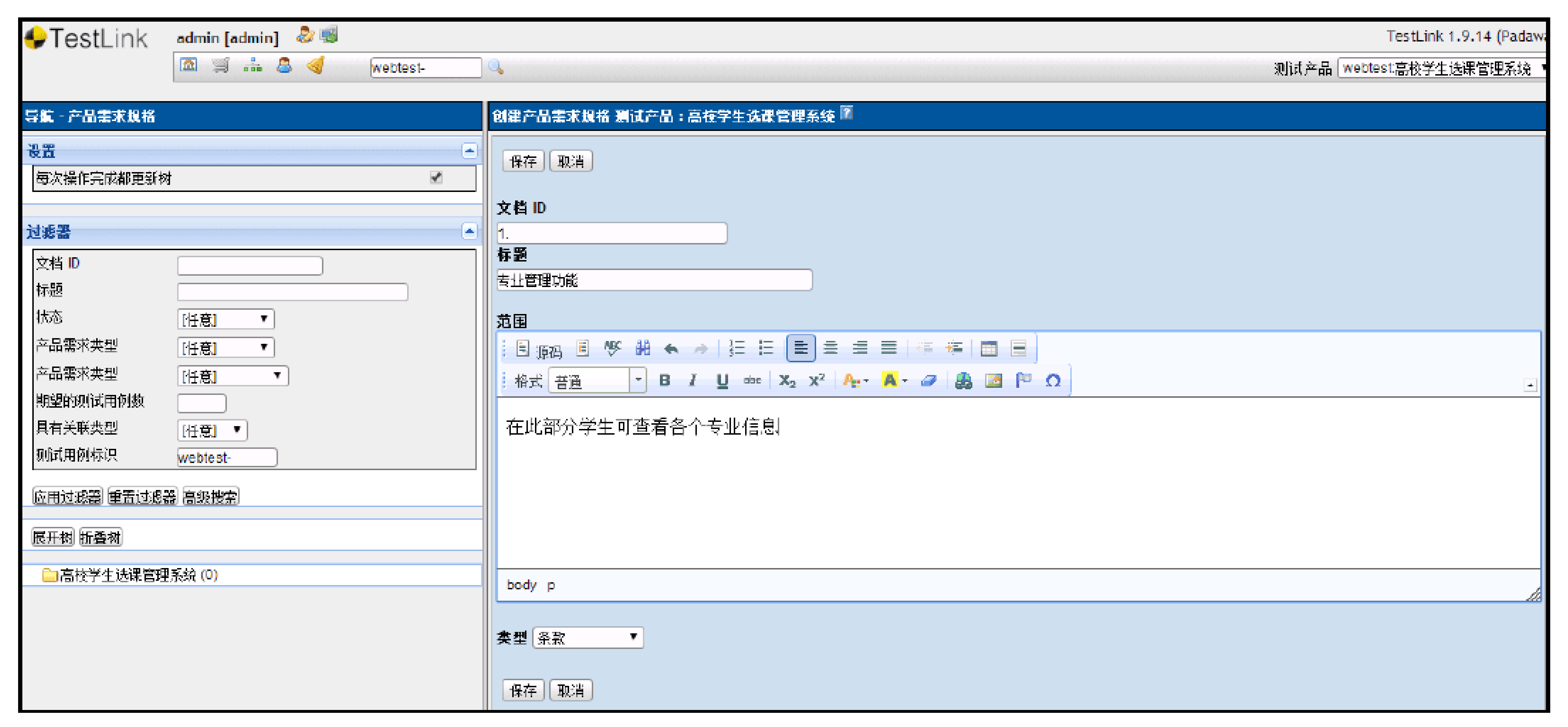


图 6-24　创建产品需求规格

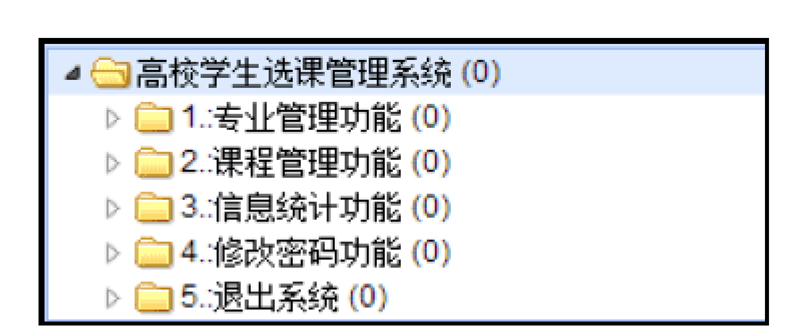


图 6-25　产品需求规格

2) 创建需求

选择要编辑的需求规格，单击该页面上的“创建新产品需求”按钮，开始新建测试需求。单击需求规格“1：前台功能测试”，添加它的需求“1.1：登录验证”，需要的测试用例数一个，如图 6-26 所示。

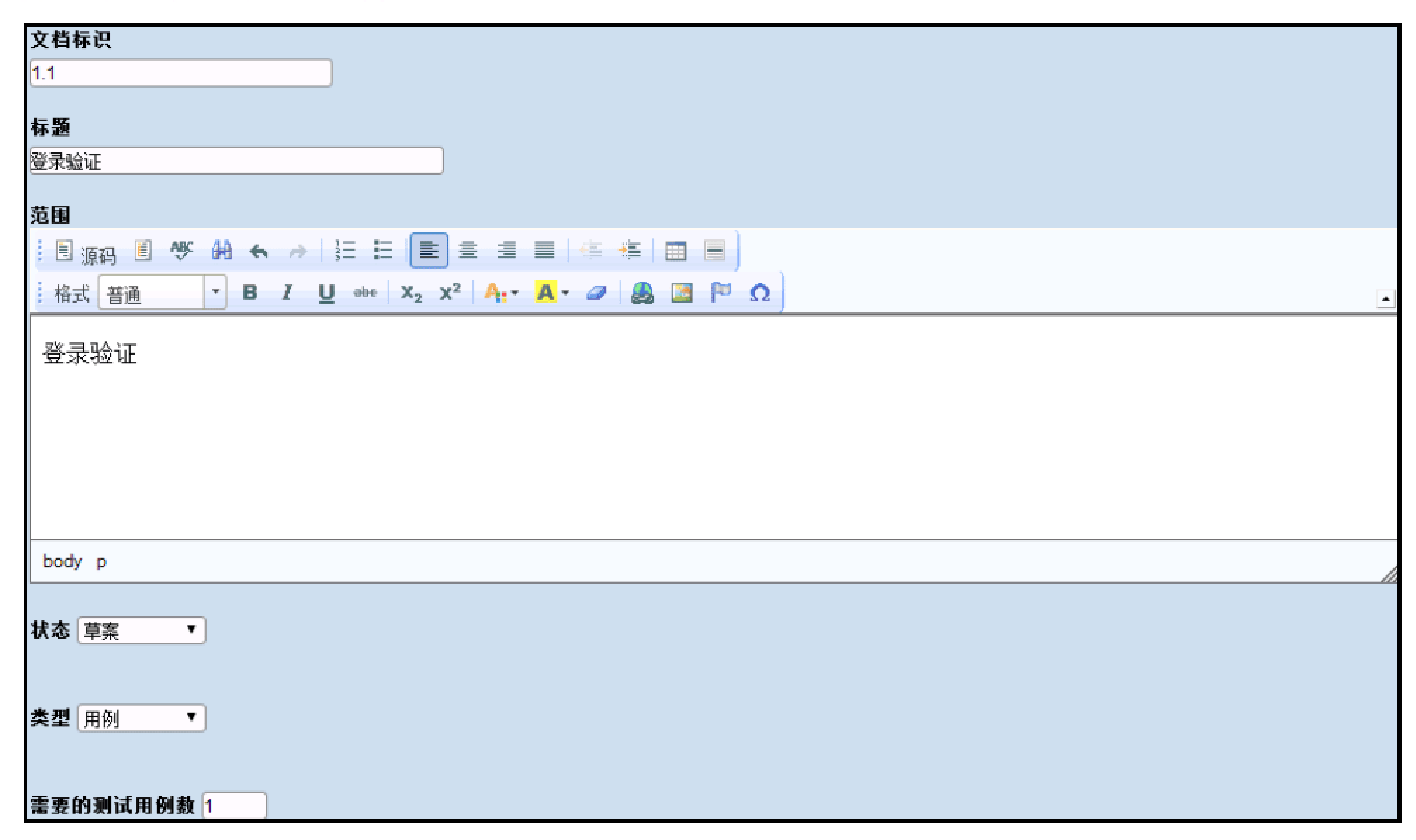


图 6-26　创建需求

添加其他需求，如表 6-5 所示。

表 6-5　所有需求

需　　求	测试用例数
登录验证功能测试	1
专业管理功能测试——查询专业信息	1
专业管理功能测试——更改专业状态	1
专业管理功能测试——增加新专业	1
课程管理功能测试——查询课程信息	1
课程管理功能测试——添加新课程	1
信息统计功能测试——查询统计信息	1
信息统计功能测试——查询学员名单	1
修改密码功能测试——修改密码	1
退出系统功能测试——退出系统	1

完成后的需求如图 6-27 所示。

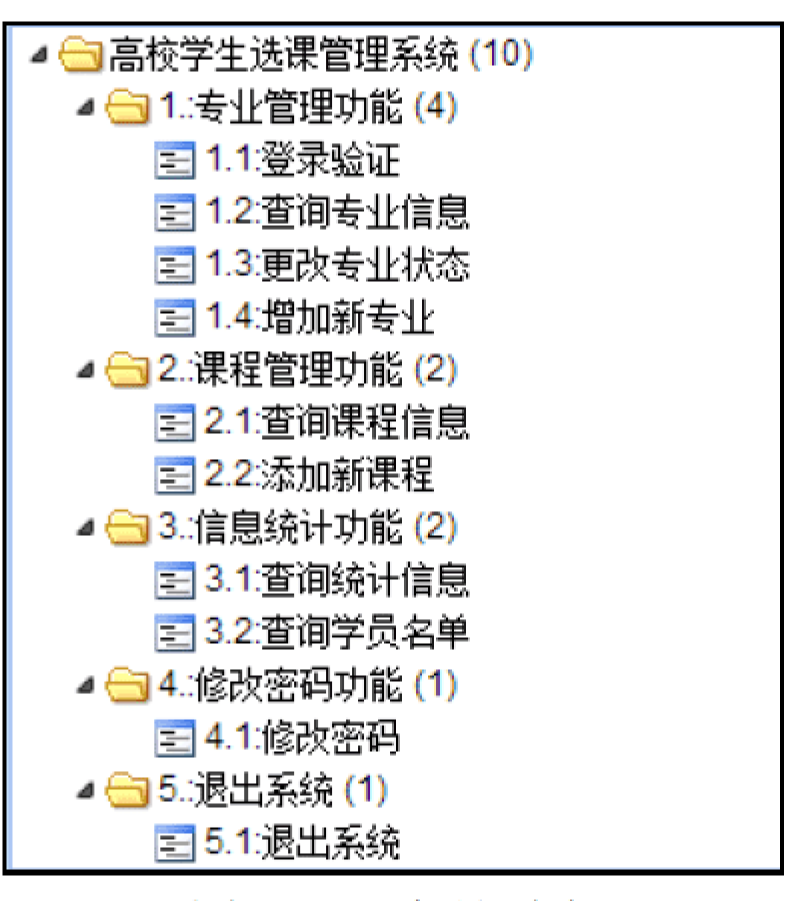


图 6-27　产品需求

测试需求内容包含：文档标识、名称、范围、需求的状态，以及覆盖需求的案例。TestLink 提供了两种状态来管理需求：合法的(valid)和不可测试的(not testable)。

TestLink 提供了从文件导入测试需求的功能，支持的文件类型有 CSV、CSV(DOOR)和 XML 三种。同时 TestLink 也提供了将需求导出的功能，支持的文件类型有 CSV 和 XML 两种。

TestLink 还提供了上传文件的功能，可以在创建测试需求的时候，为该需求附上相关的文档。

3. 测试用例管理

TestLink 支持的测试用例管理包含两层：测试用例集或测试套件(Test Suite)、测试用例(Test Case)。可以把测试用例集对应到项目的功能模块。测试用例与每个模块的功能相对应。

可以使用测试用例搜索功能从不同的项目以及成百上千个测试用例中查到需要的测试用例，甚至可以直接将别的项目里的测试用例复制过来，这样就解决了测试用例的管理和重用问题。但是，还有一个问题没有解决，那就是与测试需求的对应问题。在测试用例

管理中，测试用例对测试需求的覆盖率是人们非常关心的，从需求规格说明书中提取出测试需求之后，TestLink 会提供管理测试需求与测试用例间对应关系的功能。

1) 创建测试用例集

单击主页上的“测试用例”菜单中的“编辑测试用例”，出现如图 6-28 所示界面。

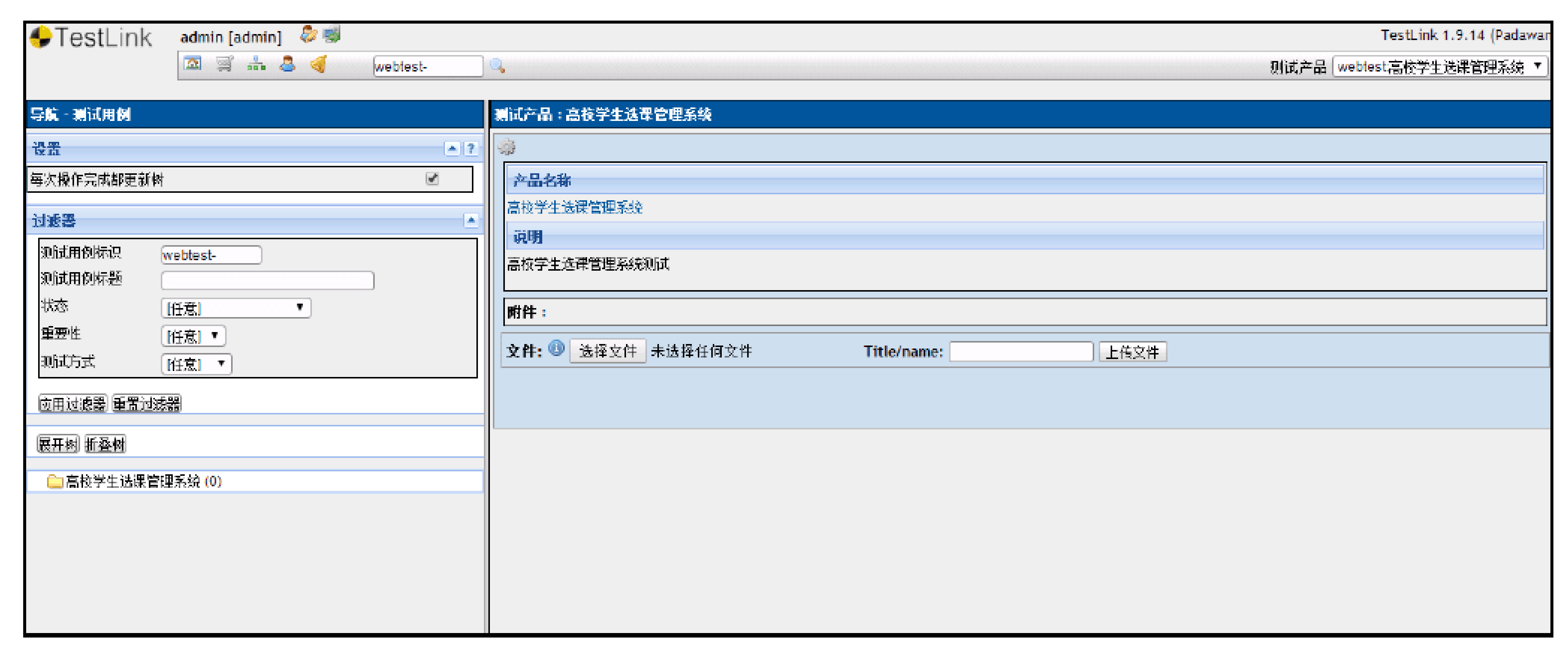


图 6-28　创建测试用例集主页面

单击左侧的“高校学生选课管理系统”，在右侧窗口中单击“新建测试用例集”按钮，添加两个测试用例集“专业管理功能”和“后台功能测试(教师端)”，如图 6-29 所示。

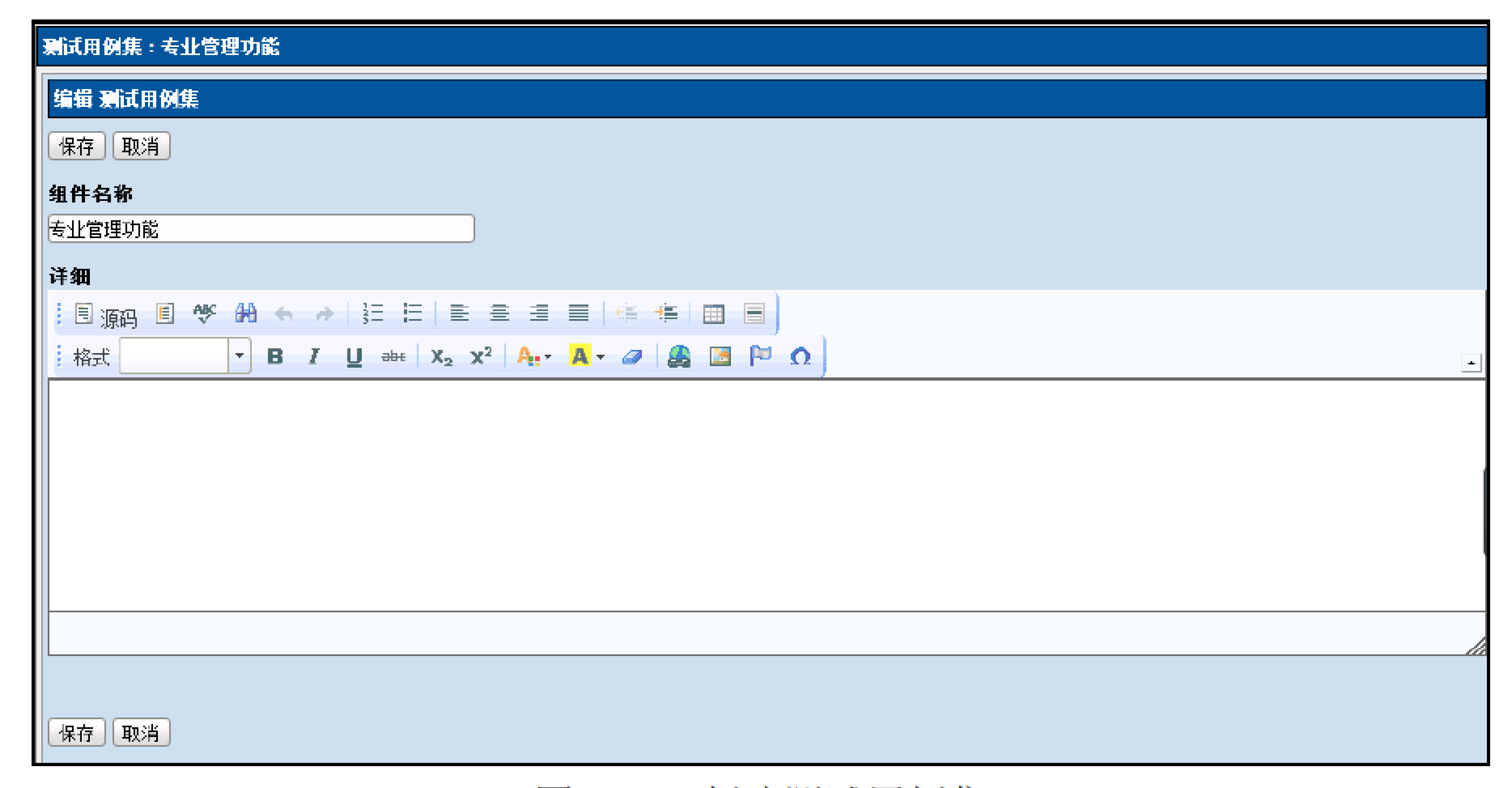


图 6-29　创建测试用例集

填写好相关内容后，可单击“创建测试用例集”按钮，创建该测试用例集，完成以后如图 6-30 所示。

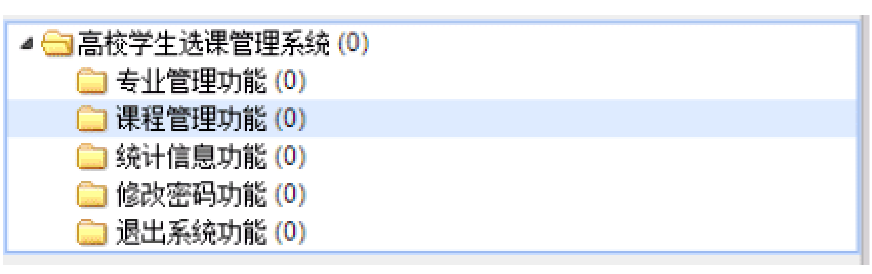


图 6-30　测试用例集创建完成

2) 添加测试用例

选择创建好的测试用例集“前台功能测试”(学生端)，单击该页面右侧的“创建测试用例”按钮，新建一个名为“登录验证”的测试用例，如图 6-31 所示。

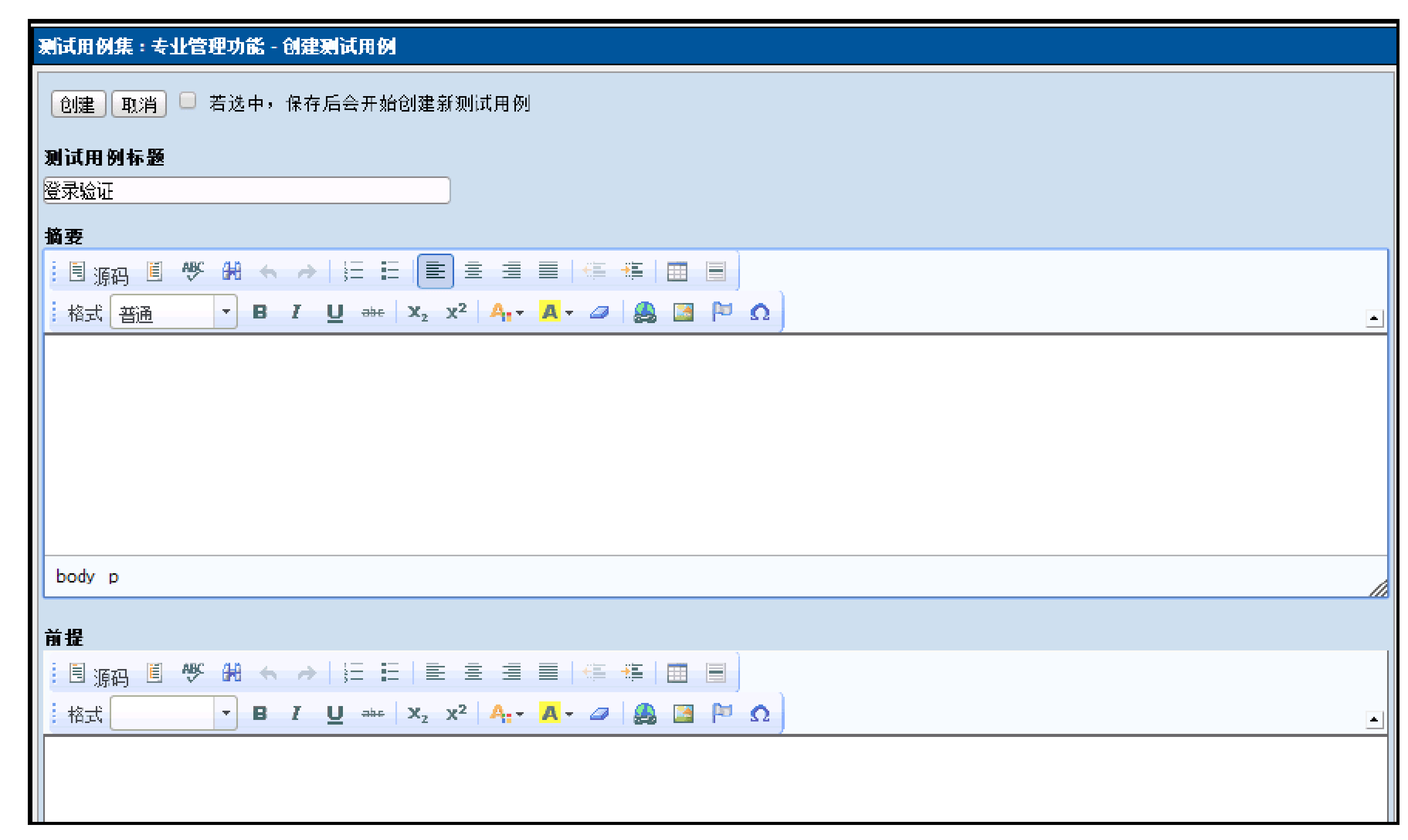


图 6-31　创建测试用例

测试用例创建成功后，单击“创建步骤”按钮，输入数据，单击“保存”按钮，如图 6-32 所示。



图 6-32　创建测试步骤

按照此方法添加如表 6-6 所示的数据。

表 6-6　部分测试数据

测试用例	步 骤 动 作	期望的结果
登录验证	1. 登录用户名为空，密码不为空	1. 提示用户名或密码不能为空
	2. 登录用户名不为空，密码为空	2. 提示用户名或密码不能为空
	3. 输入错误的用户名或密码	3. 提示用户名或密码错误
	4. 输入正确的用户名和密码	4. 显示登录成功的界面

(续表)

测试用例	步 骤 动 作	期望的结果
专业选课	1. 打开专业管理界面，单击“增加新专业”按钮 2. 在添加新专业界面，正确选择入学年份、正确输入专业名称、正确选择学制年份、单击“确定”按钮 3. 在添加新专业界面，错误选择入学年份、错误输入专业名称、错误选择学制年份、单击“确定”按钮 4. 在专业清单页面，正确选择专业名称，在操作列中设置已结业	1. 页面跳转，进入添加新专业界面 2. 页面跳转，进入专业管理界面，在专业清单中显示专业名称 3. 页面不跳转，在输入错误框上方显示红色错误提示信息 4. 操作列显示已结业
选课查询	1. 单击“查询成绩”按钮	1. 在页面上显示学生已选课程
课程管理	1. 打开高校学生选课系统主界面，单击“课程管理”按钮 2. 在“请输入搜索条件”框内，选择某专业，在“授课教师姓名”框内输入教师姓名，在“课程名称”框中输入课程名称，单击“搜索”按钮 3. 打开课程管理界面，单击“大学英语 1” 4. 单击“增加新课程”按钮 5. 增加新课程后，单击“重置”按钮 6. 打开课程管理界面，选择“学生可选”，单击课程名称 7. 打开课程详细信息页面，在“课程是否可选”处，单击“设置为不可选”	1. 页面跳转到课程管理界面 2. 页面跳转到符合条件的课程清单 3. 页面跳转到大学英语 1 的课程详细信息界面 4. 页面跳转到添加新课程界面 5. 页面跳转到添加新课程界面 6. 页面跳转到课程详细信息界面 7. 页面中“课程是否可选”行显示为“否”

测试用例包含如下部分：标题，要求是简短的描述语或缩写词(如 TL-USER-LOGIN)；摘要，要求短小、概括全面；步骤，描述测试说明(输入行为)以及可以包括的前提条件及清除信息；期望的结果，描述检验点和测试产品或系统的预期行为。

建议：在编写测试用例时，要细分每一种数据类型。有些测试用例的编写步骤是相同的，变化的可能只是数据类型，可以采用复制的方法来实现。如果有多个分类下面的测试用例操作相同，而只是部分数据类型或字段名称不同，则可以通过移动测试用例的方法来减少测试用例工作量。同时，也可以在创建测试用例的摘要中，将不同的测试数据罗列，然后在测试步骤中，根据不同的测试数据，执行相同的操作。

完成上述操作后，按照测试用例树创建测试用例，如图 6-33 所示。

3) 删除测试用例

经主管许可后，通过“删除”按钮可以将测试用例集和测试用例从测试计划中删除。当第一次创建测试计划时，由于其中还没有包含任何结论，删除数据也许是有益的。但是在大部分情况下，删除测试用例会导致与它们相关联的所有结果都丢失。因此，使用这项功能时一定要十分谨慎。



图 6-33　测试用例树

4) 需求关联

测试用例和软件/系统需求之间是 n 对 n 的关系。对于一个产品来说，这个功能是一定要用到的。用户可以通过在主页面上指派产品需求，把需求指派给测试用例。

单击主页面“产品需求”模块下的“指派产品需求”菜单，进入指派需求页面，选中左侧用例树中的测试用例，再选择右侧对应的测试需求，进行指派即可。本特性允许在需求和测试用例之间建立关系，设计人员可以定义 0..n 到 0..n 的关系。例如，一个需求可以被指派给零个、一个或多个测试用例，反之亦然。

高校学生选课管理系统中测试用例与需求的关系如表 6-7 所示。

表 6-7　测试用例与需求的关系

<table>
<tr><th>测 试 用 例</th><th>需　　求</th></tr>
<tr><td>登录验证</td><td>登录验证</td></tr>
<tr><td>更改专业状态</td><td rowspan="3">专业管理</td></tr>
<tr><td>增加新专业</td></tr>
<tr><td>查询专业信息</td></tr>
<tr><td>查询课程信息</td><td rowspan="2">课程管理</td></tr>
<tr><td>增加新课程</td></tr>
<tr><td>查询统计信息</td><td rowspan="2">统计信息功能</td></tr>
<tr><td>查询学员名单</td></tr>
<tr><td>修改密码</td><td>修改密码</td></tr>
<tr><td>退出系统</td><td>退出系统</td></tr>
</table>

完成上述操作后，可以查看已经指派的测试用例。单击顶部的产品需求栏，在所示页面中单击左下的需求“1.3：更改专业信息”，可以看到需求的覆盖率，如图 6-34 所示。

图 6-34　需求覆盖率

4. 制定测试计划

制定测试计划只能由 admin 用户进行。

1) 创建测试计划

测试计划是执行测试用例的基础，测试计划由测试用例组成，而这些测试用例是在特定的时间段里输入到产品中的。测试计划只能由主管创建，但也可以从其他测试计划中产生，同时还允许用户在紧急情况下及时地从测试用例中创建测试计划。当为一个小模块创建一个测试计划时，上述都是可以使用的方法。为了使用户可以查看测试计划，用户必须有足够的权限，而查看测试计划的权限要在用户/产品权限定义页面中由主管分配，当用户被告知他们不能查看工作中的项目时，权限是需要记住的重要事情。

在 TestLink 系统中，一个完整的测试计划要包括各测试阶段(如集成测试阶段、系统测试阶段)的名称。

单击主页面“测试计划管理”模块下的“测试计划管理”菜单。在出现的页面中单击“创建”按钮，进入测试计划创建页面，如图 6-35 所示。

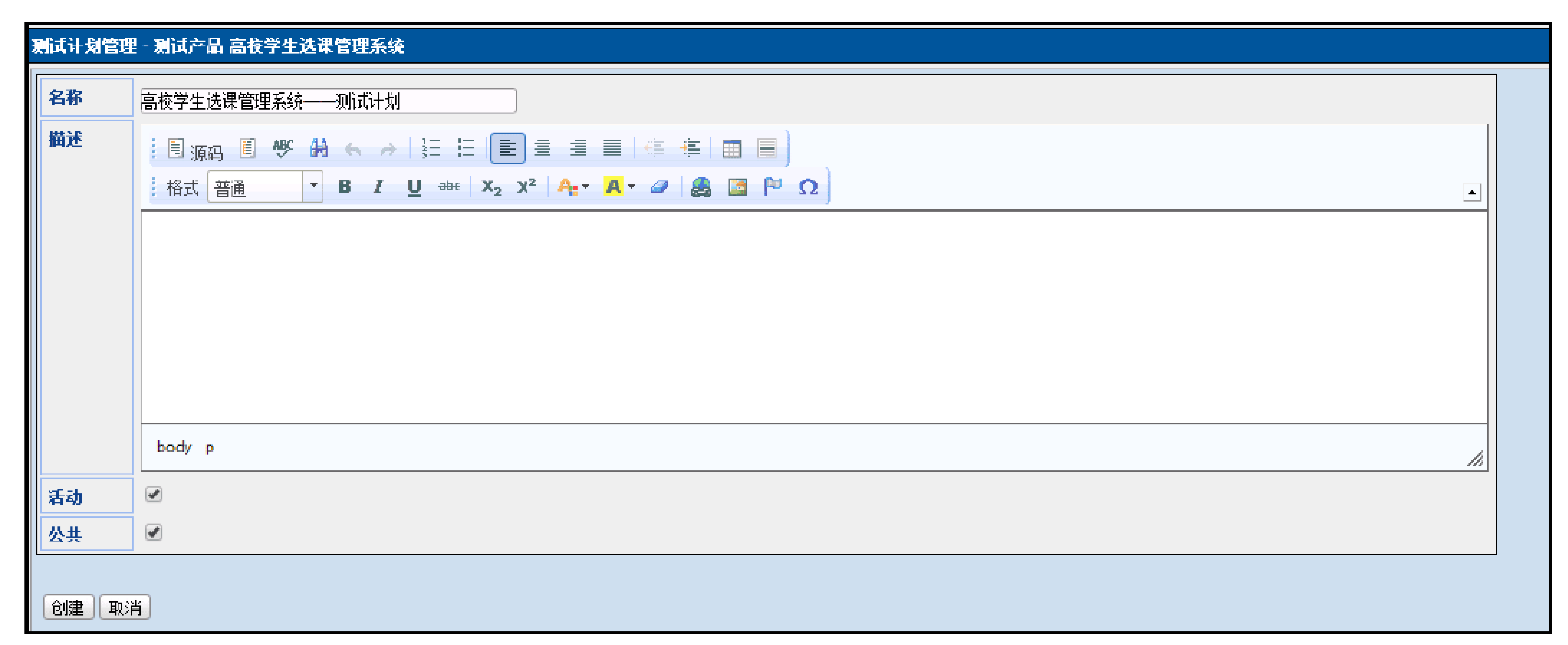


图 6-35　创建测试计划

测试计划的内容包括：计划名称、计划描述，以及是否从已有的测试计划创建。如果

选择从已有的测试计划创建，则新创建的测试计划包含选择的已有测试计划的所有关联信息，比如已有测试计划分配的测试用例。

创建一个名为“高校学生管理系统——测试计划”的测试计划。

2) 创建测试里程碑(明确每个测试阶段的开始和截止时间，以及完成 A、B、C 三种优先级的比例)

单击主页面“测试计划管理”模块下的“编辑/删除里程碑”菜单，创建一个新的测试里程碑。测试里程碑的内容包括名称、日期、优先级，如图 6-36 所示。

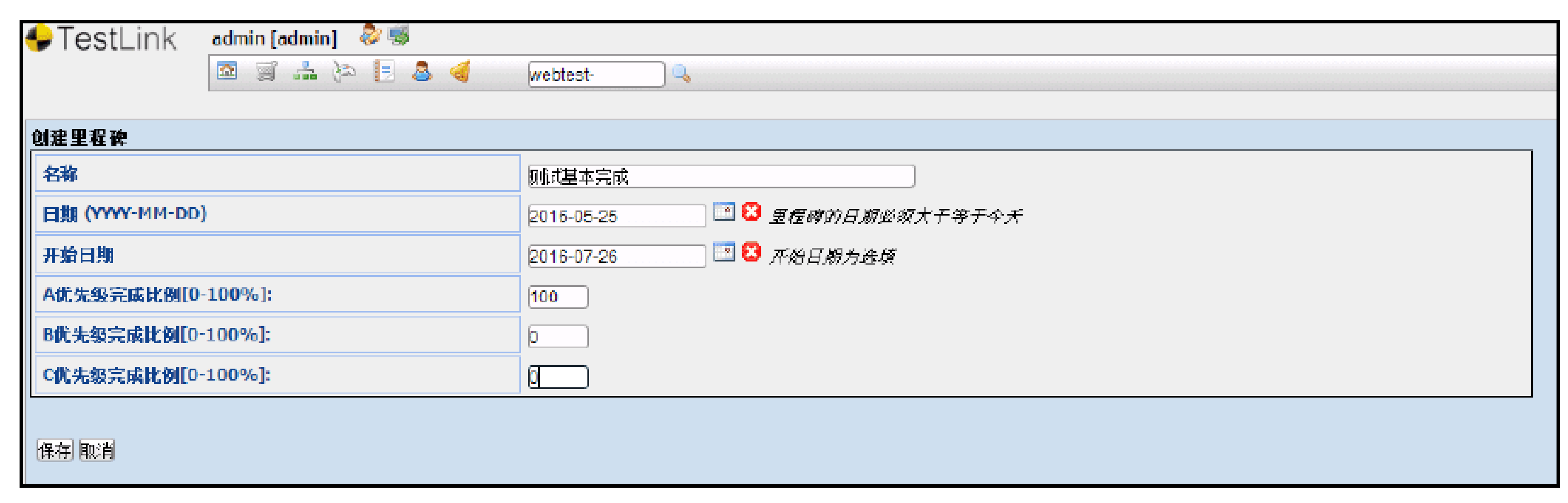


图 6-36　创建测试计划里程碑

3) 版本管理(Builds/Releases)

测试计划做好后，就应该制定版本，比如 ver 1.0。如果测试过程中发现了 bug，修改之后就产生了 ver 2.0。这时应该追加版本，相应地接下来未完的测试以及降级测试都应该在新的版本上完成。所有测试完成后，可以统计在各个版本上测试了哪些用例，以及在每个版本上是否都进行了降级测试等。

单击主页面“测试计划管理”模块下的“版本管理”菜单，创建一个新的测试版本，如图 6-37 所示。

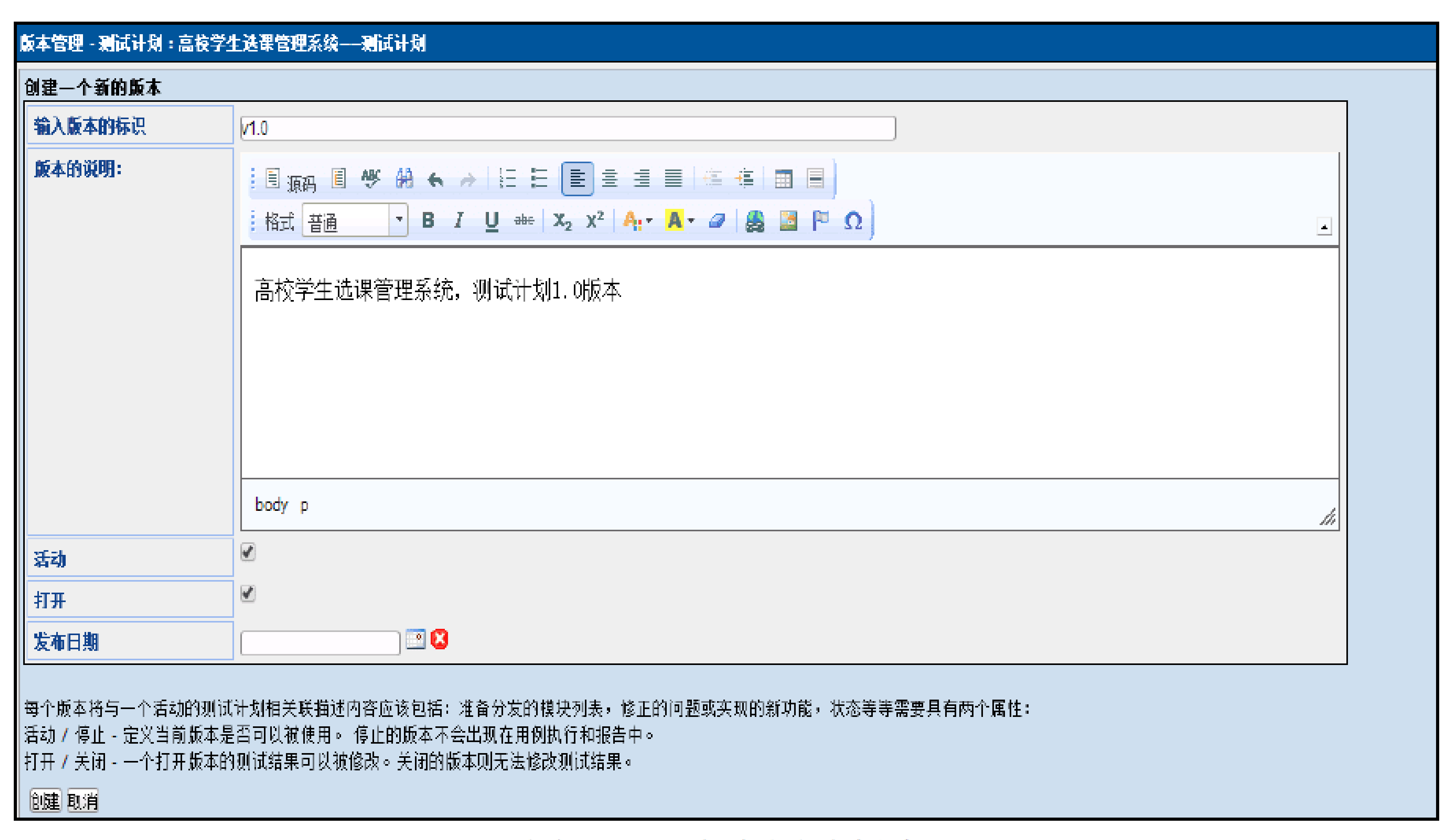


图 6-37　定义测试版本

测试版本的内容包括版本的标识、版本的说明以及选择该版本是否为活动和打开的版本。如果是活动的版本，说明该版本可以用，否则说明该版本不可用。如果是打开的版本，该版本的测试结果可修改，否则不可修改。在 TestLink 中，执行由版本和测试用例组成。

如果在一个项目中没有创建版本，执行页面将不允许执行，度量页面则完全是空白的，版本通常不能被编辑或删除。

4) 安排测试人员(定义用户/测试计划的角色及权限)

单击主页面“测试计划管理”模块下的“指派用户角色”菜单，为测试计划指派用户。测试用户如表 6-8 所示。

表 6-8　测试用户

账　号	用户类型	备　注
Tester1	Tester	测试工程师 1
Tester2	Tester	测试工程师 2
TD1	TdESIGNER	测试组长
ST1	Senior Tester	资深测试工程师
Test_manager1	Leader	测试经理

在为测试计划指派用户页面上，可以选择测试计划，选择好需要指派权限的测试角色后，单击“更改”按钮，可以更改测试计划。选择好测试计划后，可以将该测试计划以不同的角色分配给不同的用户，通过角色列表，可以选择用户对该测试计划的操作角色，选择结束后，单击“更新”按钮，可以保存结果，如图 6-38 所示。



图 6-38　指派测试计划用户角色

5) 添加测试用例到测试计划中

在主页面通过测试计划下拉列表，先选择一个测试计划，单击“测试用例集”下的“添加/删除测试用例到测试计划”按钮，进入在测试计划中添加测试用例界面。

可以将已经创建好的测试用例指派给该测试计划。单击一个测试用例集，可以看到该测试用例集下所有的测试用例。可以选择该测试计划中要执行的测试用例，单击“增加选择的测试用例”来添加或删除测试用例，可以将选择好的测试用例分配给该测试计划，如图 6-39 所示。



图 6-39　添加测试用例到测试计划中

6) 移除测试用例

在如图 6-39 所示的页面中，可以在复选框组中勾选不需要在该测试计划中执行的测试用例，然后单击“添加/删除选择的”按钮，将测试用例移除，如图 6-40 所示。

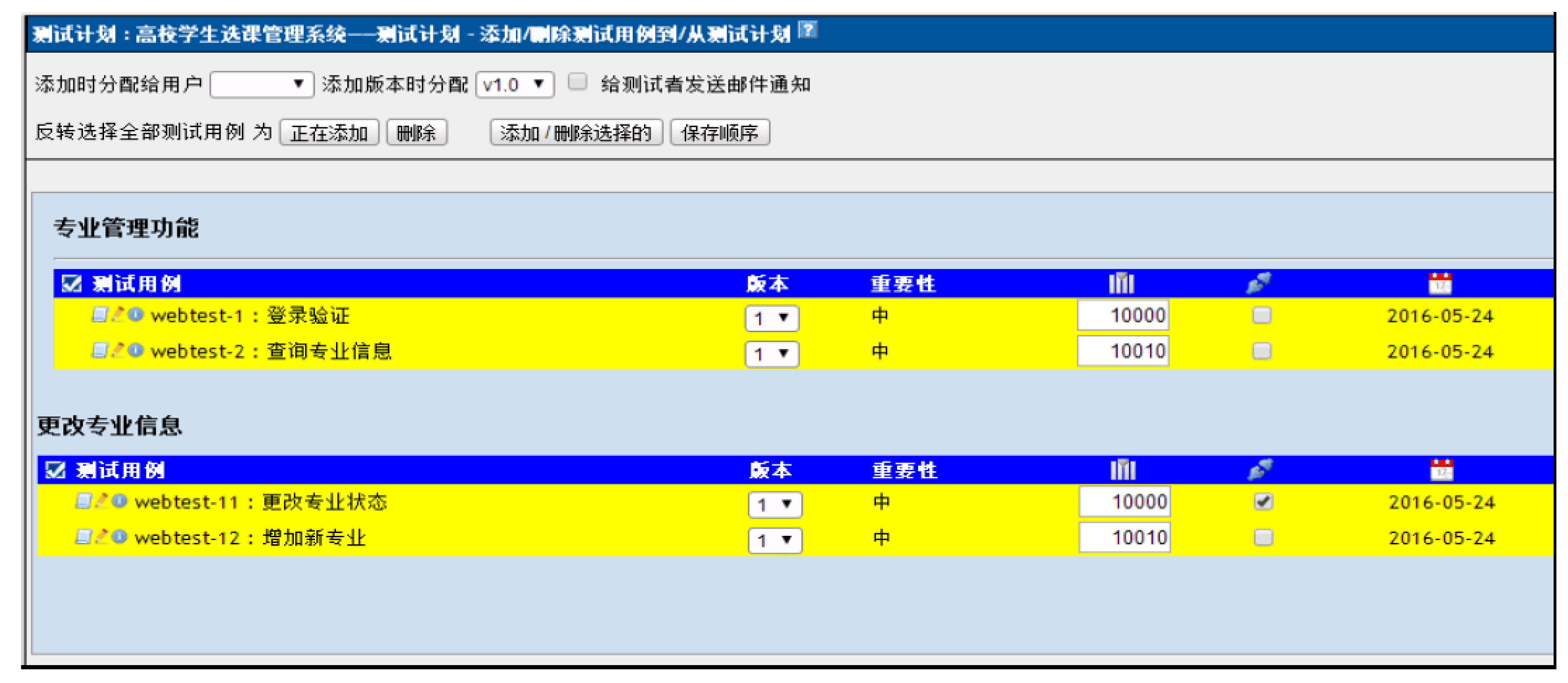


图 6-40　移除测试用例

7) 设置测试用例的所用者(给测试人员分派测试任务)

单击主页面“测试用例集”模块下的“指派执行测试用例”菜单，进入指派测试用例页面，可以为当前测试计划中所包含的每个测试用例指定一个具体的执行人员。

在指派测试用例页面，从左侧用例树中选择某个测试用例集或测试用例，右侧页面上将会出现下拉列表以供选择用户。选择好合适的用户后，选中测试用例前面的复选框，单击“保存”按钮即可完成测试用例的指派工作，如图 6-41 所示。

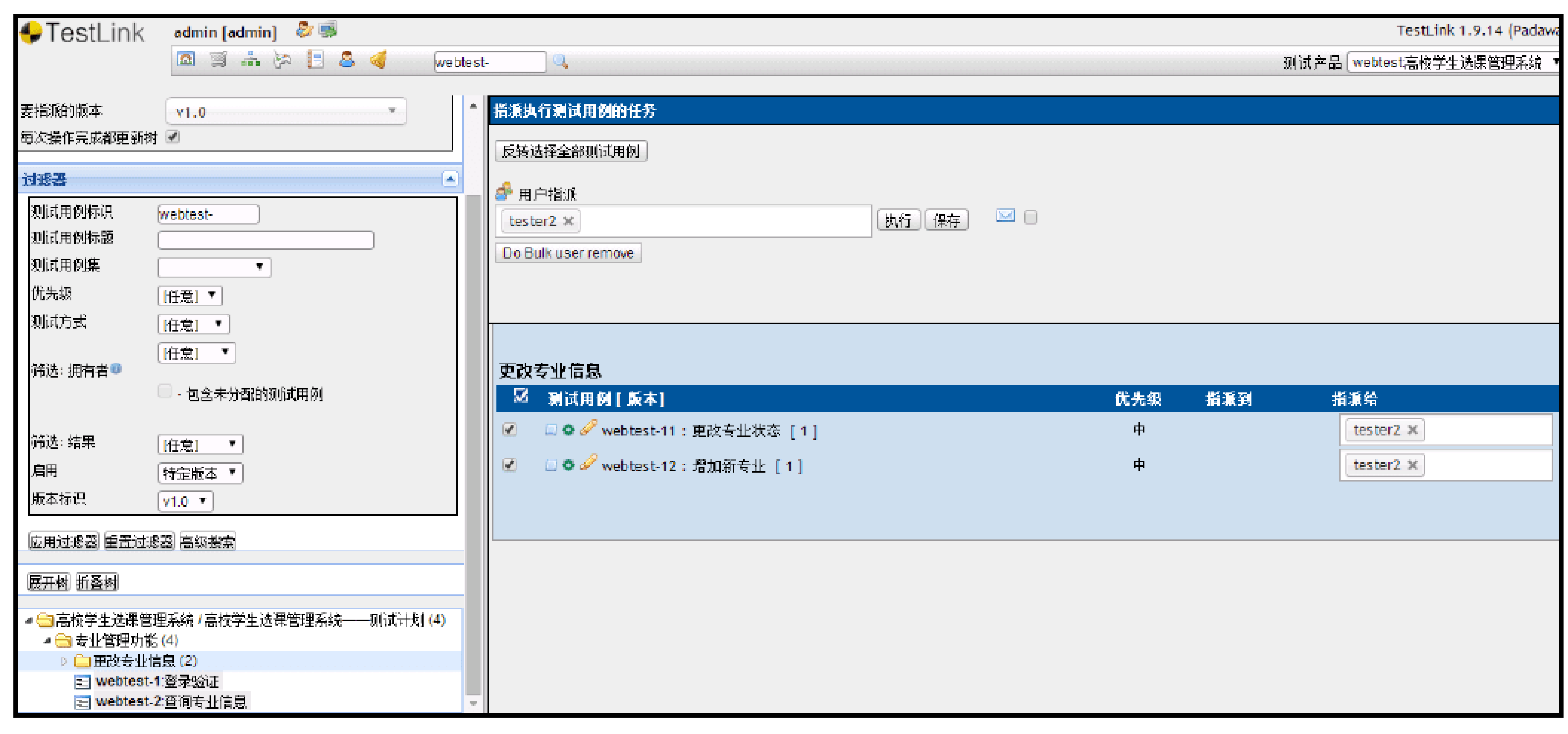


图 6-41　设置测试用例的所用者

当然，在这里也可以进行批量指定——右侧页面的最上方有一个下拉列表可以选择用户，在下方的测试用例列表中选择要指派给该用户的用例，然后单击后面的“执行”按钮即可完成将多个用例指派给一个人的操作。

8) 执行测试

在 TestLink 顶部的菜单栏中有“执行”选项，单击进入测试用例执行页面，如图 6-42 所示。

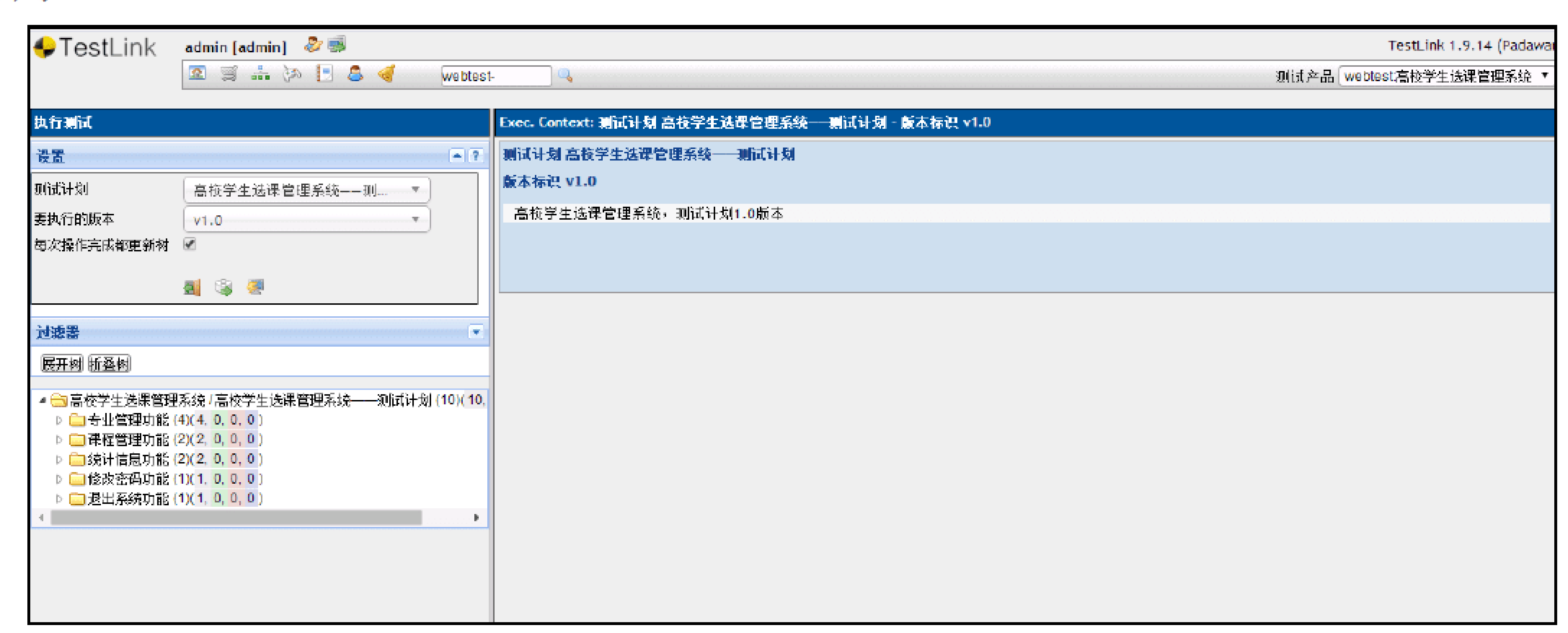


图 6-42　测试用例执行页面

这里需要说明一点，虽然执行表面上针对的是测试计划，但实际上对应的是测试计划中测试用例的执行情况。

在左侧的用例树中，可以根据具体的条件来选择测试用例。选择某个测试用例集后，页面右侧上方会出现测试计划、build 描述、测试集说明等信息，页面下方则是每个测试用例的详细情况，同时在每一个测试用例的最后部分，有“说明/描述”对话框，可以在这里输入执行的一些说明性情况，还有测试“结果”对话框，这两个对话框都需要执行完测试用例后自己填写。

测试结果分为以下 4 种情况：

(1) 通过：该测试用例执行通过。

(2) 失败：该测试用例没有执行成功，这个时候可能就要向 Mantis 提交 bug 了。

(3) 锁定：由于其他用例执行失败，导致此用例无法执行，被阻塞。

(4) 尚未执行：如果某个测试用例没有执行，则在最后的度量中将其标记为“尚未执行”。

进行测试的页面如图 6-43 所示。

摘要

前提

#	步骤动作	期望的结果	执行	Execution notes	Result
1	1.登录用户名为空，密码不为空 2.登录用户名不为空，密码为空 3.输入错误的用户名和密码 4.输入正确的用户名和密码	1.提示用户名和密码不能为空 2.提示用户名和密码不能为空 3.提示用户名和密码错误 4.显示登录成功的界面	手工		

文件：选择文件 未选择任何文件

测试方式：手工

Estimated exec. duration (min) :

说明/描述

结果

Execution duration (min)

保存结果　保存并进入下一个

图 6-43　进行测试

5. 测试报告

执行测试用例的过程中一旦发现 bug，需要立即把其报告到 bug 管理系统 Mantis 中。TestLink 提供了与多种 bug 跟踪系统关联的接口配置，目前支持的 bug 管理系统有 Jira、Bugzilla、Mantis。与 Mantis 集成的方法如下：

(1) 主页→System→Issue Tracker Management，单击“创建”，添加一个 Issue Tracker，如图 6-44 所示(单击 Show configuration example 可弹出示例，如果不能弹出，到 lib/issuetrackerintegration/目录下的相关文档中找 public static function getCfgTemplate 函数)。

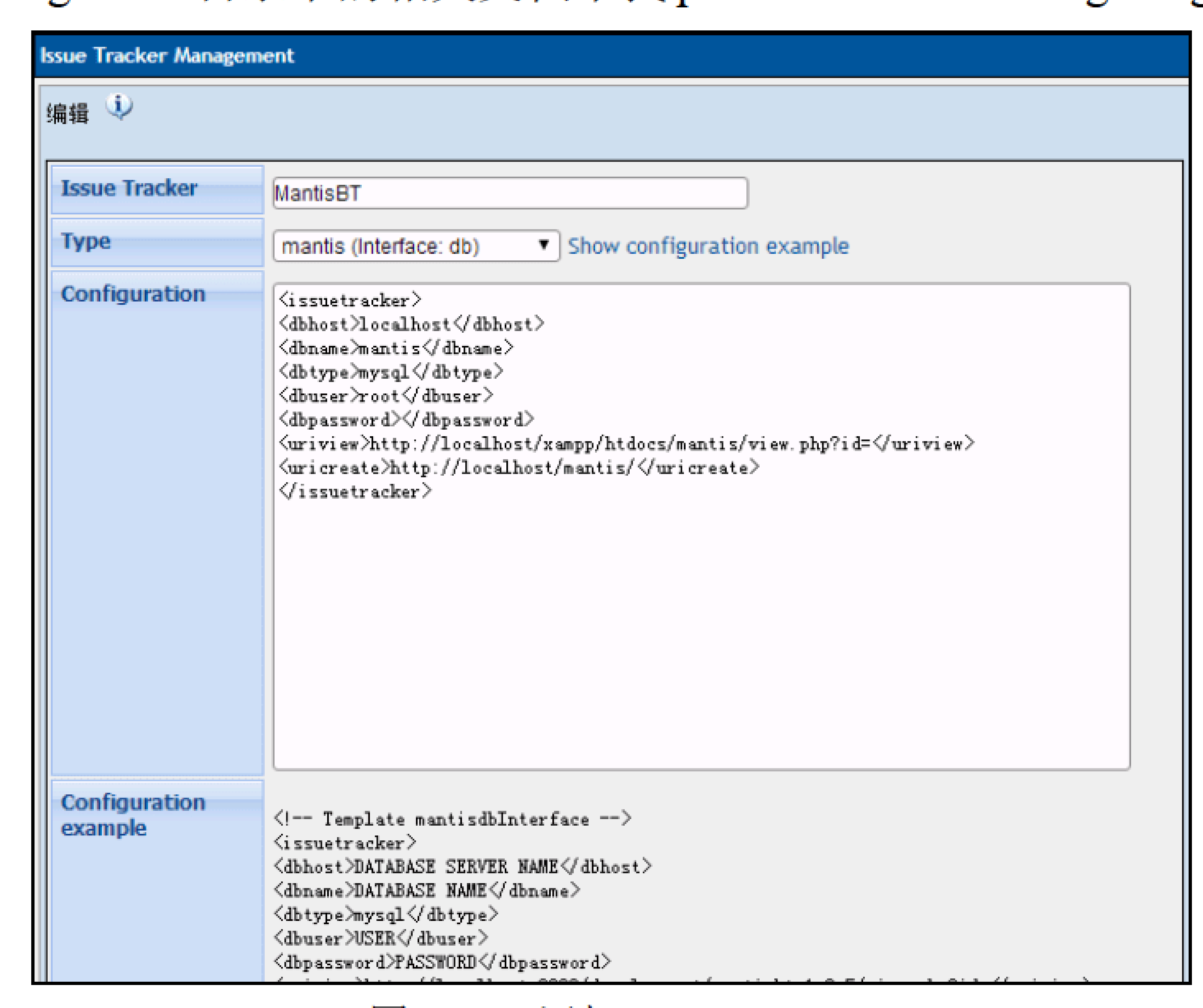


图 6-44　添加 Issue Tracker

(2) 主页→产品管理→测试项目管理，在项目编辑页面选择缺陷跟踪系统，如图 6-45 所示。

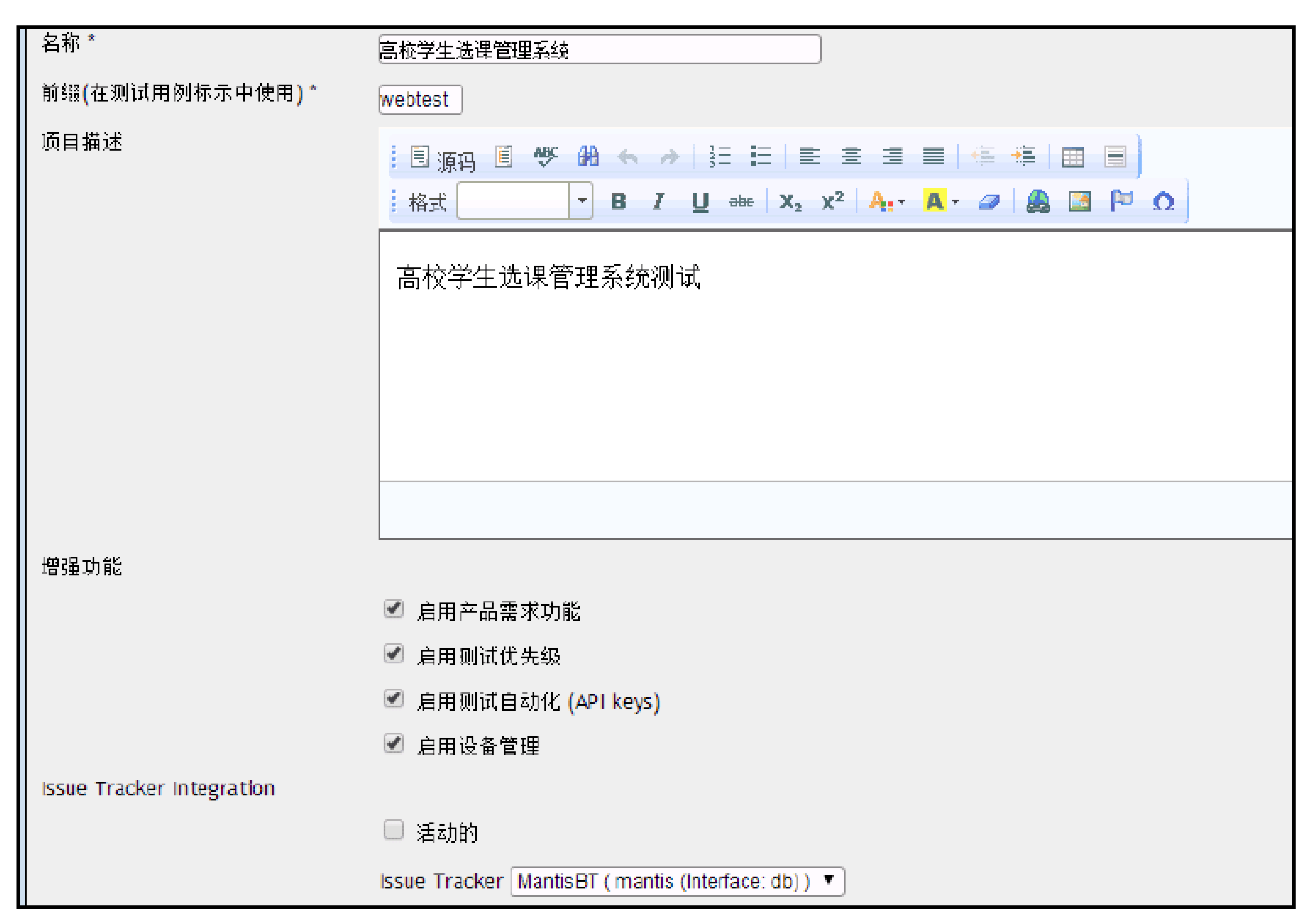


图 6-45　选择缺陷跟踪系

(3) TestLink 和 Mantis 集成后，执行测试，测试结果中会多出一项用于 bug 管理的项，在这个记录的后面会有一个类似于小虫子的标记，单击这个小虫子标记后，会出现一个记录 bug 编号的输入框，如图 6-46 所示。如果测试用例是失败的，可以在这个地方输入该测试用例所发现 bug 在 Mantis 中的 ID，然后在该记录的下面会出现一个 ID 链接，如图 6-47 所示。单击该 ID 链接后，可以直接链接到 Mantis 中该 bug 的页面，如图 6-48 所示。



图 6-46　添加 bug



图 6-47　与 Manits 关联的链接

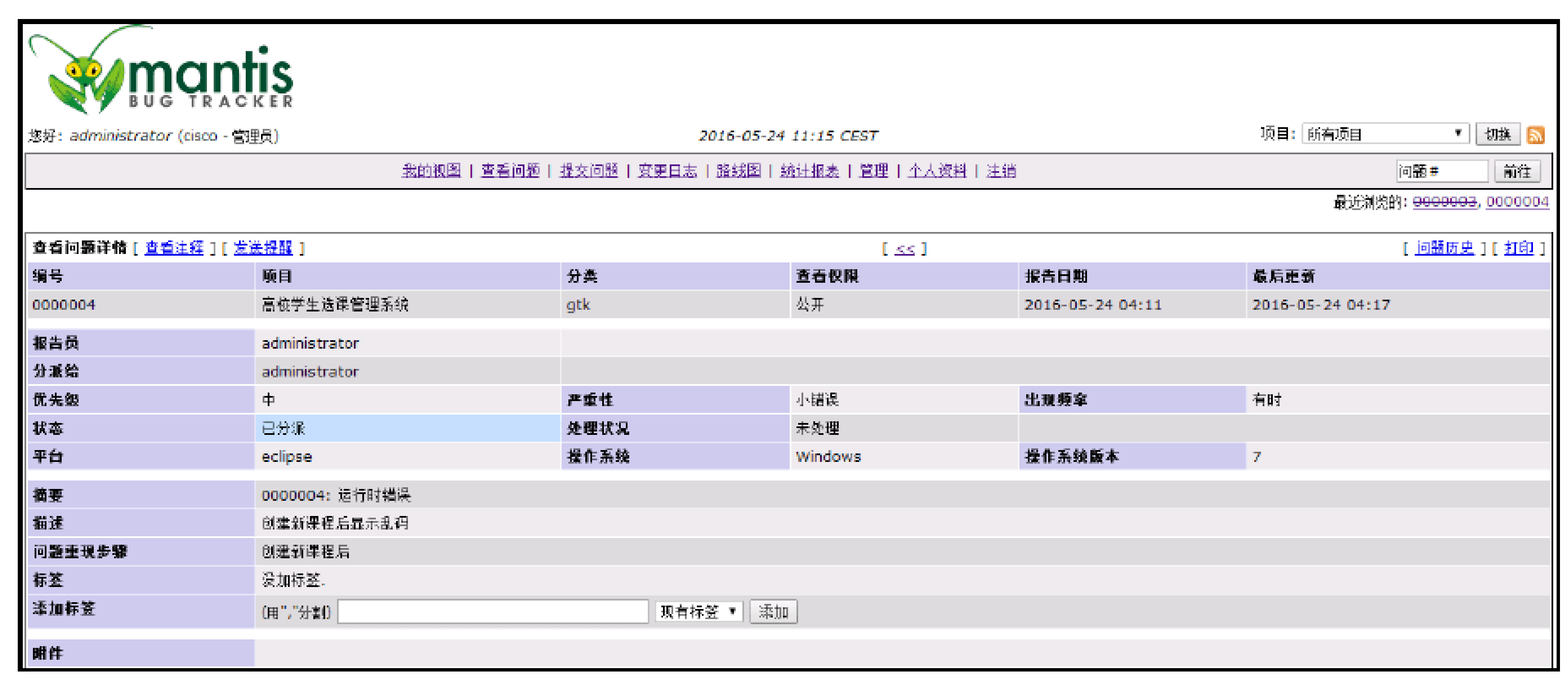


图 6-48　Mantis 中的 bug 页

6. 分析测试结果

TestLink 根据测试过程中记录的数据，提供了较为丰富的度量统计功能，可以直观地得到测试管理过程中需要进行分析和总结的数据。单击首页导航菜单栏中的“结果”菜单，即可进入测试结果报告页面，主要包括的功能如图 6-49 所示。

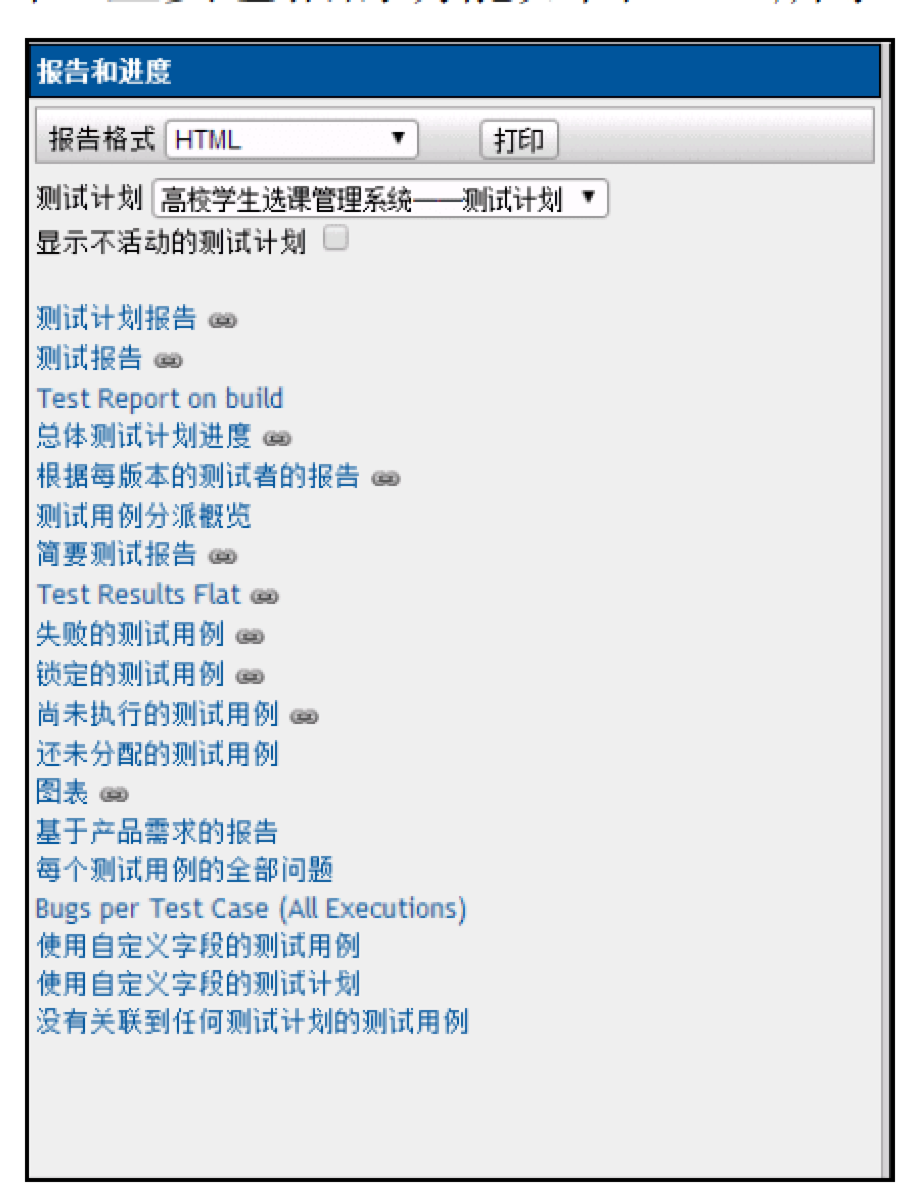


图 6-49　测试结果报告主要包括的功能

1) 总体测试计划进度

查看总体的测试情况，可以根据测试组件、测试用例拥有者、关键字进行查看，如图 6-50 所示。

2) 失败的测试用例

统计所有当前测试结果为失败的测试用例。

3) 锁定的测试用例

统计所有当前测试结果为锁定的测试用例。



测试产品：高校学生选课管理系统
测试计划：高校学生选课管理系统——测试计划

- 每个版本的测试结果进度

版本标识	指派的	尚未执行	[%]	通过	[%]	失败	[%]	锁定	[%]	已完成 [%]
v1.0	10	0	0.0	8	80.0	2	20.0	0	0.0	100.0

- 每个版本的测试结果进度 日期是根据版本分派了测试人员的测试用例计算得出。

上级测试用例集的测试结果

组件	总数	尚未执行	[%]	通过	[%]	失败	[%]	锁定	[%]	已完成 [%]
专业管理功能	4	0	0.0	3	75.0	1	25.0	0	0.0	100.0
修改密码功能	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0
统计信息功能	2	0	0.0	2	100.0	0	0.0	0	0.0	100.0
课程管理功能	2	0	0.0	1	50.0	1	50.0	0	0.0	100.0
退出系统功能	1	0	0.0	1	100.0	0	0.0	0	0.0	100.0

此报告显示所有顶级测试用例集的结果。子测试用例集的结果也被考虑在内。

根据测试用例的优先级显示测试结果

优先级	总数	尚未执行	[%]	通过	[%]	失败	[%]	锁定	[%]	已完成 [%]
中	10	0	0.0	8	80.0	2	20.0	0	0.0	100.0

图 6-50　总体测试计划进度

4) 尚未执行的测试用例

统计所有当前尚未执行测试的测试用例，如图 6-51 所示。



没有运行测试用例

测试产品：高校学生选课管理系统
测试计划：高校学生选课管理系统——测试计划

There are no test case (WITH TESTER ASSIGNED) that have not been run

图 6-51　尚未执行的测试用例

5) 图表

单击“图表”，可以看到 TestLink 以图表形式生成的报告，非常直观。这里主要是通过图表的形式来表示测试用例的执行情况，红色表示测试失败，蓝色表示锁定测试用例，绿色表示通过测试，黑色表示尚未执行，如图 6-52 和图 6-53 所示。

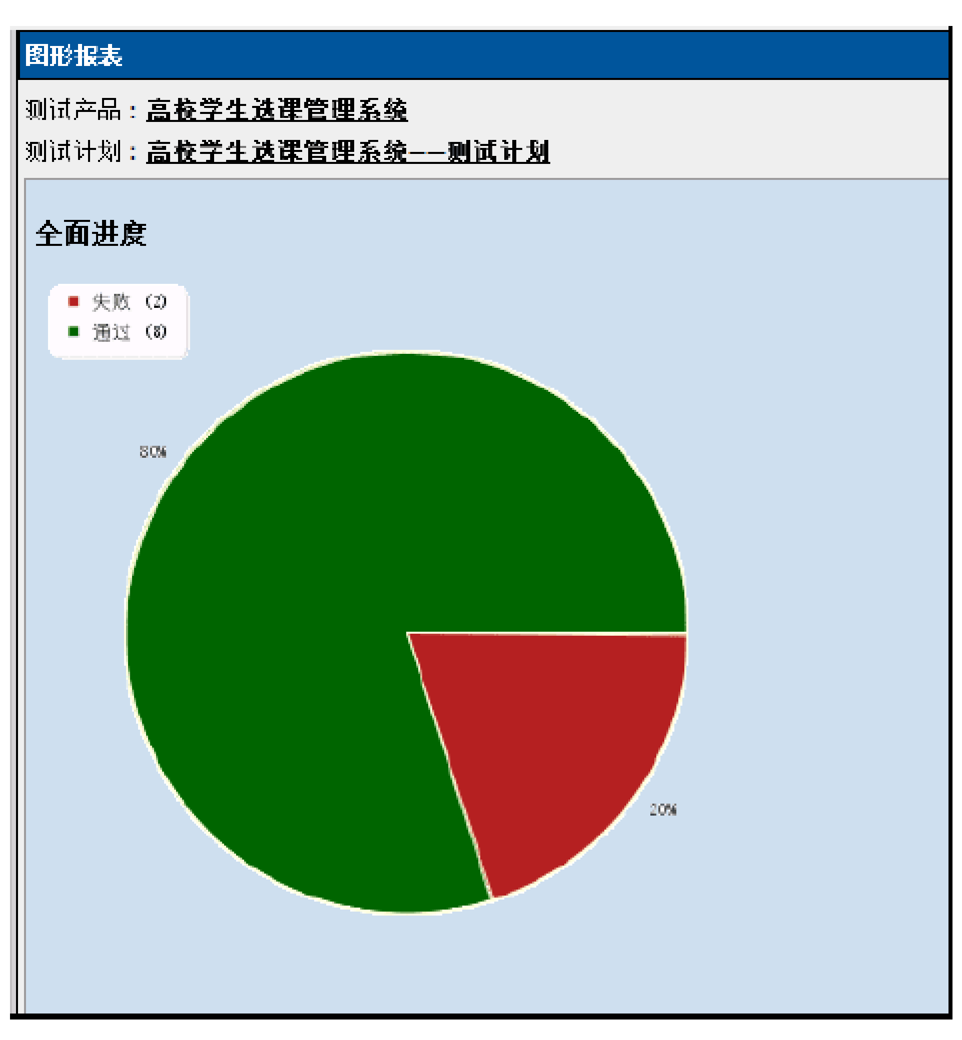


图 6-52　总体测试结果饼图

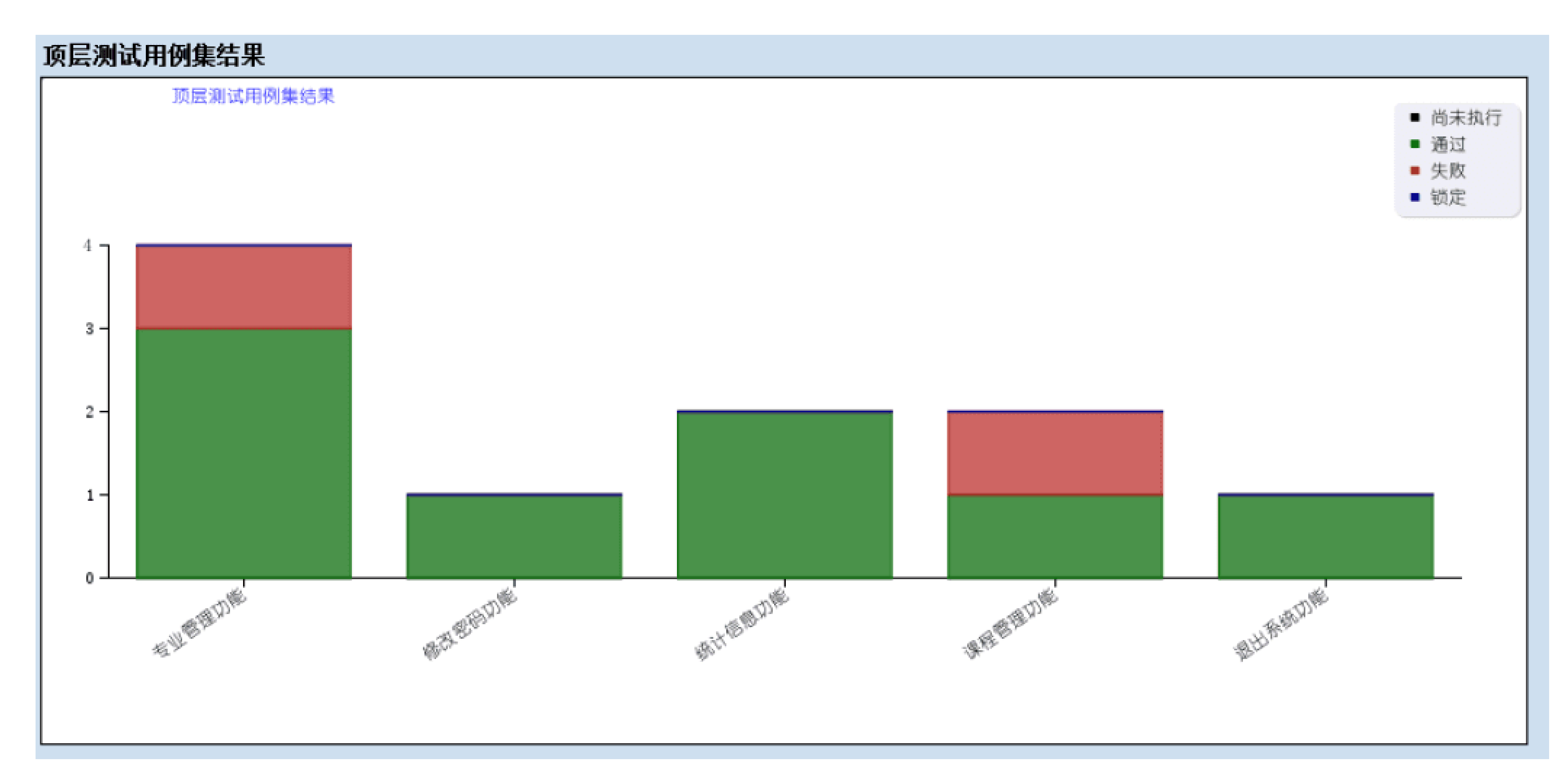


图 6-53　顶层测试用例集结果

在 TestLink 图表中，显示乱码的解决办法如下：

(1) 从 Windows 字体库中找到 SIMYOU.TTF(幼圆)，将其复制到 TestLink 中的 pchat 字体目录，路径为 E:\xampp\htdocs\testlink\third_party\pchart\Fonts(以本书安装路径为例)。

(2) 修改 config.inc.php 文件，将$tlCfg->charts_font_path =TL_ABS_PATH."third_party/pchart/Fonts/tahoma.ttf";中的字体 tahoma.ttf 修改为 SIMYOU.TTF。

总的来说，TestLink 已经创造了很好的测试管理条件，其配置也相对简单，可以根据自己的需要进行相关的配置。另外，TestLink 是开源的测试工具，这也提高了灵活性。不过在 TestLink 的使用过程中，用户也感到了一些不便，比如在很多情况下，需要回到主页面才能单击一些链接，并且 TestLink 和缺陷管理工具的整合需要手工来完成。另外，一些描述信息需要用文字来描述，如果能提供填表的方式，效果会更好。

习题和思考题

1. 软件测试过程模型主要有哪些？它们之间有什么关系？各表明什么意思？
2. 如何在软件测试中运用软件测试过程模型？运用中要注意些什么问题？
3. 简述软件测试过程的概念，软件测试过程包含哪些活动和内容？我们怎样对软件测试过程进行度量？
4. 什么是软件测试过程成熟度？CMM 与软件管理工具之间有什么关系？软件测试过程改进与软件过程改进存在什么样的关系？
5. 简述软件测试管理的流程，软件测试过程管理包括哪些基本内容？
6. 软件测试管理各阶段要完成的主要任务有哪些？需要编写哪些文档？
7. 我们怎样获取软件测试需求？在软件测试设计中，需要考虑哪些问题？

8. 我们怎样执行软件测试用例、分析软件测试结果、撰写软件测试文档？

9. 简述软件测试用例、测试数据与测试脚本三者的概念以及它们之间的关系。

10. 简述软件测试过程中的配置管理及组织管理的思想、方法和技术手段。

11. 建立 TestRunner(Testopia)测试管理环境，尝试与相关的缺陷管理工具集成，并与 TestLink 进行功能比较。

12. 建立一个测试管理与缺陷管理的环境，并通过一个具体实例来完整地说明测试管理与缺陷管理的流程。

第Ⅲ部分　软件测试方法与技术篇

从第Ⅰ部分和第Ⅱ部分的内容可以知道无论是传统的软件测试还是基于生命周期的软件测试，最终的目的就是发现软件中的各种错误，确保软件的质量，检验被测软件是否满足规定的需求。尽管我们在软件测试生命周期中采用单元测试、集成测试、配置项测试(也称软件合格性测试或确认测试)、系统测试、验收测试和回归测试等测试类别或测试级别，并对被测软件进行功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余物测试及强度测试等各种种类的测试，然而软件测试并不能保证发现和修复被测软件中所有的错误。因此，怎样使得软件测试在资源和进度允许的条件下尽可能多地发现软件中存在的错误，从而使软件测试达到最佳的测试效果，就是本部分要介绍的内容，即从软件测试方法(又称为软件测试技术)的角度论述这个问题的解决答案。这些软件测试方法的科学使用将是软件测试任务高效率、高质量完成，软件质量得到充分保证，软件需求得到全面满足的技术保障。

软件测试是一项实践性很强的工作，它的发展是一个从实践到理论，又从理论回到实践而不断往复的过程。而且随着软件开发技术、软件编程方法的发展，软件系统的不断扩大，结构越来越复杂，应用面越来越广，致使软件测试技术和测试方法也在持续发展。目前，软件测试方法主要分为静态测试和动态测试两大类，静态测试和动态测试下面又包含很多测试方法或分为很多子类，如静态测试包括代码审查、代码走查、桌面检查、技术评审和静态分析等，动态测试则包括黑盒测试、白盒测试以及灰盒测试等。

采用何种测试方法并如何使用该测试方法进行软件测试是测试阶段的关键技术问题。所谓测试方法，包括具体的测试目的(例如，预定要测试的具体功能)、应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例。其中最困难的问题是设计测试用例的输入数据。

不同的测试数据发现程序错误的能力差别很大，为了提高测试效率、降低测试成本，应选择高效的测试数据。因为不可能进行穷尽测试，所以选用少量最有效的测试数据，做到尽可能完备的测试就更重要了。

设计测试方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。已经研究出来的众多测试方法，各有优缺点，不能说哪一种最好，哪一种最不好，更没有哪一种可以替代其他的方法，完成测试；同一种测试方法在不同的环境下应用，得到的效果也是不一样的，因此通常都是多种测试方法联合使用，以达到最佳测试目的。

第7章　软件静态测试

静态测试通常是指不执行程序代码而寻找代码中可能存在的错误或评估程序代码的过程。被测对象是各种与软件相关的有必要进行测试的产物，例如各类文档、源代码等。静态测试可以手工进行，也可以借助软件工具自动进行。静态测试具有以下特点：

(1) 静态测试不必动态地运行程序，也就是不必进行测试用例的设计和结果分析等工作。

(2) 静态测试可以人工进行，充分发挥人的思维优势。在发现错误的同时也就可以定位错误。俗话说“解铃还须系铃人”，由于人的思维的局限性以及交流之间的障碍所造成的逻辑错误，由人通过逻辑思维去解决，是一种行之有效的方法，特别是在使得人的思维优势互补得到充分发挥后，测试的水平就会很高。

(3) 静态测试不需要特别的条件，容易展开。这是根据前个特点而得出的。

(4) 静态测试对测试人员要求较高，至少测试人员要具有编程经验。

之所以要进行静态测试，是因为一个软件可能暂时实现了需求说明书中的所有要求，但是由于它的内部结构复杂、混乱，代码的编写也没有规范，使得软件内部存在一些不易被察觉的错误，这些错误在特定的条件下会造成重大的影响；另外虽然软件目前基本完成了用户的要求，但是随着时间的推移，软件产品需要升级维护，由于程序的复杂度、代码编写的杂乱造成软件的维护工作很难进行。静态分析所要做的就是对代码标准以及质量进行监控，以此来提高代码的可靠性，使系统的设计符合模块化、结构化、面向对象的要求。静态分析主要是通过对源代码扫描或检查的方法来发现软件中的缺陷，为日后的维护工作节省大量的人力、物力，从而更有效地保证软件的质量。

静态测试主要包括各阶段评审、代码检查、程序分析、软件质量度量等。

7.1　各阶段评审

评审是对软件元素或项目状态进行评估的活动，用以确定与预期结果之间的偏差和相应的改进意见，通常由人来执行。除了在项目早期发现缺陷和降低项目失败风险外，项目中需要进行评审的其他原因包括：分享知识、培训团队成员、为管理层决策提供依据、为过程改进提供信息以及项目所处状态评审。一般评审包括：培训评审、预备评审、同行评审，我们所关心的是同行评审。另外，需求阶段的规格说明书也是评审的重要内容。

7.1.1　同行评审

同行评审是由开发软件产品的作者以外的其他人检查工作产品，以发现缺陷并寻找改进的机会。评审方法是评审参与者通常采用一行一行仔细阅读被评审对象的形式发现被测

对象中的缺陷。评审的时间点一般设在里程碑附近，即当工作产品到达一个完成的里程碑并即将进入下一个开发阶段时，如图 7-1 所示。

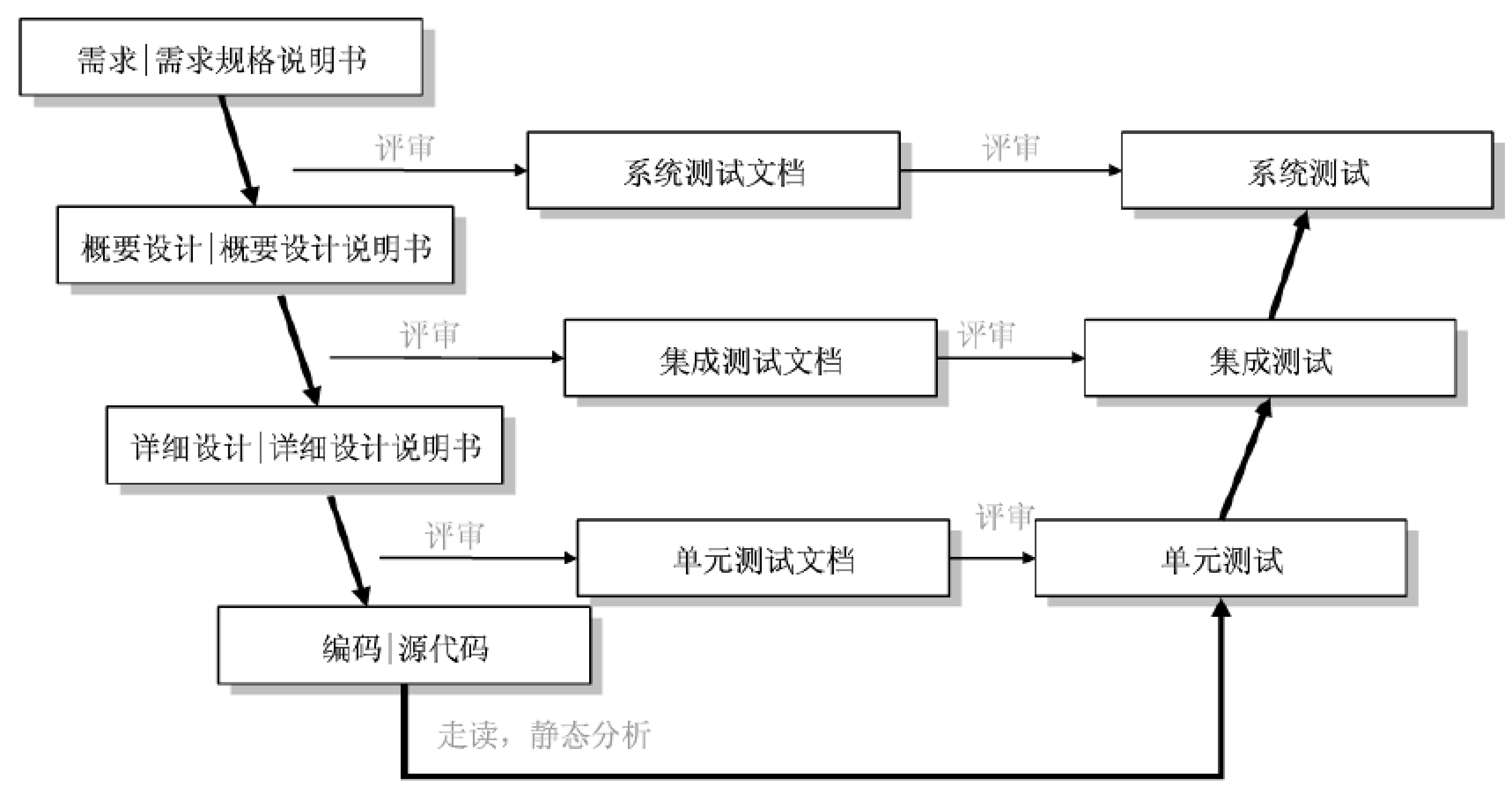


图 7-1　评审的时间点设置

同行评审一般包括审查、小组评审、走查、桌面评审、临时评审五种类型。

这些同行评审类型的区别在于正式程度：

① 审查最正式，然后是小组评审、走查、桌面评审，临时评审最随意；

② 同行评审越正式，发现的缺陷越多，但评审越正式，花费成本越高；

③ 被评审对象越重要或风险越高，采用的评审方式就越正式。

1. 审查

审查的概念是 IBM 的工程师 Michael Fagan 于 20 世纪 70 年代提出的，也叫正式评审，是一种包括非作者等专家在内的针对特定对象(如需求规格说明书、设计文档和源代码)进行检查以发现缺陷的过程。

审查是一种有结构、有规则的评审方法。Fagan的审查流程包括：计划、介绍会议、准备、会议、返工、跟踪、因果分析。每个阶段需要确定的内容：参与审查的角色，相应的输入/输出。图 7-2 给出了审查流程示意图。

1) 审查中的角色

(1) 作者(被评审对象的创建者，提供被评审对象及相关信息)。

(2) 评审组长(组织评审会议，确保审查活动能够正确地进行)。

(3) 审查专家(发现被评审对象中的问题)。

(4) 读者(在会议上讲解被评审对象，使评审专家把精力集中在被评审对象本身而不是作者)。

(5) 记录员(记录会议阶段有价值的信息)。

2) 审查工作流程

(1) 计划(参与者有作者和评审组长)。在这个阶段，需要开展下面这些工作：选择评审组长，确定审查对象，确定审查专家，确定总体会议、会议次数和相应的时间表，准备和分发审查工作包(审查包中包括被审查对象的初始可交付产品、相关参考文档、缺陷检查表、指导书、错误记录模板和其他材料)。

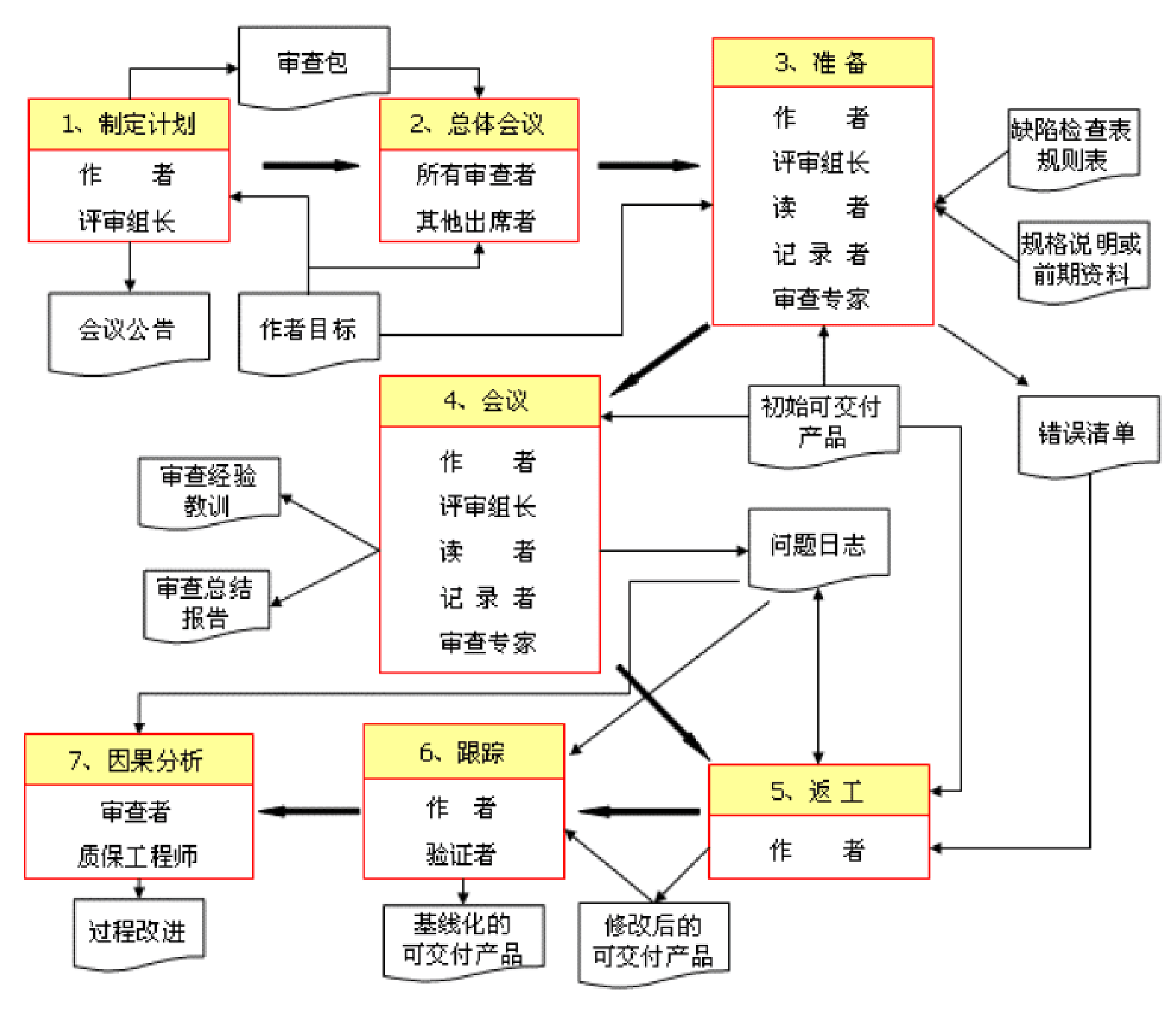


图 7-2　审查流程示意图

(2) 总体会议。本阶段是可选的，主要目标是让审查专家熟悉被审查对象，包括对象特征、上下文、背景等。参与者可以是所有需要参加审查的人员。

(3) 准备(参与者主要是审查专家，这是审查最重要的阶段)。在这个阶段，审查专家独立工作、逐行阅读被审查对象，将任何发现的问题、疑问记录在审查意见单中，评审组长根据各个审查专家提交的意见决定是否按时或推迟召开审查会议。

(4) 会议(参与者有作者、评审组长、审查专家、读者、记录员)。在会议阶段，读者分段逐个阅读审查对象，审查专家听取讲解并考虑是否有新的问题提出。评审组长组织对所有审查意见单上的问题列表进行确认，作者确认是否问题，记录员在问题列表上记录答复以及在会议上发现的新缺陷。在会议结束前，所有人投票，给出对工作产品的审查结论。

(5) 返工(参与者主要是作者)。在此阶段，作者修改会议中确认的问题，输出修改后的交付产品。

(6) 跟踪(参与者有评审组长、质量工程师、指定的审查专家)。检查修改后的交付产品，如果通过，则输出可进入基线的交付物。

(7) 因果分析(参与者主要是质量工程师)。在这个阶段，开展分析缺陷原因、度量审查效率和效果的工作。

3) 审查规则

为了更好地发挥审查的作用，在审查中有一组需要遵守的原则：①作者不能担当评审组长、读者或记录员等角色，要保持开放的思想，接受别人的意见，避免争论；②评审组长不要同时担任记录员；③控制审查小组的规模，三到七名审查专家为好。

审查中需要遵守的原则有：①审查专家要努力发现被审查对象中的问题，审查过程中始终保持对问题的敏感性；②审查期间要努力发现问题而不要试图去解决问题；③会议限制在两个小时之内；④在会议上，审查团队要保持适当的审查速度，每小时 150 至 200 行代码或 3 至 4 页文档。

2. 小组评审

小组评审类似于审查，是一种轻型审查，可采用审查的指导方针和流程，只是没有审查正式，也没有审查严格。会议期间读者的角色由评审组长代替，小组评审方法发现问题的数量是审查的 2/3。

3. 走查和同级桌查

走查是产品的作者向一组同事说明该产品，希望获得他们的意见以满足自己的需要。走查是一种非正式的评审，其过程由作者主持，没有标准的流程可循。走查发现的缺陷数量比审查发现的缺陷数量要少一半。

同级桌查是一对一评审，是指除作者以外只有一位评审专家对工作产品进行检查。

4. 临时评审

请团队内其他同事帮忙，在短时间内解决一些问题。

5. 软件评审指导书

软件评审指导书的用途是将评审过程和规则以指导书的形式固定下来，其内容包括：目的、范围、评审角色及职责、过程准则、目标、进入标准、活动、退出标准、度量、相关资料、过程监控。

7.1.2　测试需求规格说明书

检查软件的需求规格说明书一般采用逐行阅读说明书以发现缺陷的方式，对需求规格说明书的测试应该在需求阶段规格说明书整体或部分完成后立即开展。其目的是尽早地发现缺陷，使规格说明书具有更好的可测试性，软件测试人员可以更加熟悉系统应用。所采用的具体方法是静态黑盒测试(由于规格说明书非常重要，很多软件项目评审将规格说明书作为重要的评审内容之一)。在进行规格说明书审查时可以采用对说明书进行概要评审和对说明书进行详细评审的技术。

1. 规格说明书的概要评审

对规格说明书进行概要评审所要达到的目的是发现特定的缺陷，比如大的原理性问题、遗漏或过度复杂的描述等。评审时，测试人员要站在用户的角度(确保作为第一质量要素的用户要求得到满足)，研究现有标准和基线，对类似的软件系统进行评审和测试。

(1) 站在用户的角度问自己：我需要什么样的功能？我需要的所有功能是否都包含在规格说明书中了？是否存在与现有系统冲突的功能？功能是否易于使用？性能如何？功能的安全情况如何？等等。当然，测试人员如果能从一些熟悉软件目标应用领域的人那里获得支持，对评审过程将是非常有帮助的。

(2) 当对规格说明书进行概要评审的时候，测试人员应该参考现有的标准和基线，如组织标准、术语和惯例(软件应该使用终端用户的通用术语和惯例)，工业标准(在某些工业领域，例如通信、金融，有很多应用软件必须遵守的协议)，政府标准，安全标准，等等。测试人员应该把相关标准作为规格说明书评审的一部分。

(3) 没有什么比经验更好的东西了，从类似软件中可以得到大量有用的信息，这些参

考软件可能是：正在开发系统的早期版本，组织内的类似软件，竞争对手产品。分析类似软件的时候，应密切关注相关问题并考虑这些问题是否会影响被测系统，如特性是否有增删？代码变更比例如何？软件的复杂度是否有区别？可测试性如何？性能、安全性和其他一些非功能特性如何？最后，不要忘了可以从公共出版物和网上找到有价值的信息。

2. 规格说明书的详细评审

测试人员对规格说明书进行详细评审，可从有关属性方面着手，好的规格说明书应具有如下属性：

(1) 完整性：是否忘记或遗漏了什么内容？是否彻底？是否包含了规格说明书所必须包含的信息？

(2) 精确性：建议的提案是否正确？是否定义了合适的目标？是否有什么错误？

(3) 准确性、明确而清晰：描述是否清晰明确，是否有歧义？是否易于阅读和理解？

(4) 一致性：特性描述内部和特性之间是否相互矛盾？

(5) 相关性：细分特性是否必需？是否需要去除不必要的信息？特性是否可以跟踪到原始用户需求？

(6) 可行性：项目计划和预算都是明确的，在给定的人力、工具和资源条件下，特性能否实现？

除了上述属性外，我们还可关注代码无关属性(规格说明书的目标是定义产品需求而不是软件设计、架构和代码)和可测试属性(特性是否可测试？是否提供了让测试人员得以验证功能的足够信息？)。

检查规格说明书的同时，时刻关注评审的文字和图片是否具有这样的属性。

3. 问题词语列表

测试规格说明书的时候应密切关注下面的一些词汇以及这些词汇的上下文含义是否清晰，因为这些词汇常常会带来缺陷：

(1) 总是、每个、所有、没有一个、从来不等。这些词表示肯定和确定的含义，必须确认该用这些词语或找出不该使用的理由。

(2) 当然、所以、明显地、无疑、显然等。这些词有劝说人接受的意思，在规格说明书中应尽量避免。

(3) 一些、有时、经常、通常、大部分、主要的、等等、类似、好、快、便宜、高效、小和稳定等。这些词可测试性差，必须进一步定义以给出确切的含义描述。

(4) 有把握的、处理过的、拒绝的、跳过的、去掉的等。这些词可能隐藏一些本该详细说明的功能性需求。

(5) 如果……那么……等。这些描述依赖于其他因素，不可取。

7.2　代码检查

代码检查是白盒测试的一种静态测试方法，是众多软件测试方法中发现软件缺陷最有

效的方法之一。主要由检验人员通过仔细地进行代码分析，检查代码和设计的一致性、代码对标准的遵循、代码的可读性、代码的逻辑表达正确性、代码结构的合理性。例如，一些问题是代码虽然可以正常运行，但是代码的编写不符合某种标准或规范，这就相当于写出来的东西可以被人们理解和使用，但是不符合语言的语法和文法规范。标准是建立起来的、经过修改和必须遵守的规则——做什么和不做什么，规范是建议的最佳做法，标准是必须遵守的，规范可以适当放宽。

有的代码可以运行，甚至在测试中也表现出稳定性，但是因为不符合某些标准而仍被认为是有问题的代码，造成这种现象的原因有很多，但主要因为如下三个重要原因：

(1) 可靠性。大量的事实证明按照某种标准和规范编写的代码比不这样做的代码更加可靠和安全。

(2) 可读性和可维护性。符合某种标准和规范的代码易于阅读、理解和维护。

(3) 移植性。代码经常要在不同的平台上运行或者使用不同的编译器进行编译。如果代码符合设备标准，迁移到另一个平台上就会轻而易举，甚至完全没有问题；相反，程序在编写时未考虑移植的问题，测试时也只在一个平台上进行，那么当需要迁移到另一个平台上时就要做大量的修改甚至重新编写。

在代码全部或部分完成后，应立即进行逐行代码评审以发现缺陷。以达到尽早发现缺陷，使得程序更具可测试性，软件测试人员可以更加熟悉系统的目的。

代码检查要求评审人员有程序开发语言的专业知识，有程序基线和标准供参考。

代码检查的内容有：

(1) 完整性检查(代码是否完全实现了设计文档中提出的功能需求，代码中是否存在任何没有定义或没有引用到的变量、常数或数据类型)。

(2) 一致性检查(代码的逻辑是否符合设计文档，代码中使用的格式、符号、结构等风格是否保持一致)。

(3) 正确性检查(代码是否符合制定的标准，所有的变量都被正确定义和使用，所有的注释都是准确的)。

(4) 可修改性检查(代码涉及的常量是否易于修改，如使用配置、定义为类常量、使用专门的常量类等)。

(5) 可预测性检查(代码是否具有定义良好的语法和语义，代码是否无意中陷入了死循环，代码是否避免了无穷递归)。

(6) 健壮性检查(代码是否采取措施避免运行时错误，如空指针异常等)。

(7) 可理解性检查(注释是否足够清晰地描述了每个子程序，对于没用的代码注释是否删除；是否用到不明确或不必要的复杂代码，它们是否被清楚地注释；使用一些统一的格式化技巧来增强代码的清晰度，诸如缩进、空白等；是否在定义命名规则时采用了便于记忆且反映类型的方法；循环嵌套是否太长、太深)。

(8) 可验证性检查(代码中的实现技术是否便于测试)。

(9) 结构性检查(程序的每个功能是否都作为一个可辨识的代码块存在，循环是否只有一个入口)。

(10) 可追溯性检查(代码是否对每个程序进行了唯一标识，是否有一个交叉引用的框架可以用来在代码和开发文档之间相互对应，代码是否包括一条修订历史记录，记录中对

代码的修改和原因都有记录，是否所有的安全功能都有标识)。

7.2.1　代码检查方法

所谓代码检查，其实就是以组为单位阅读代码，是一系列规程和错误检查技术的集合。该过程通常将注意力集中在发现错误上，而不是纠正错误。

代码检查一般采用静态白盒测试的方法，如代码审查、桌面检查、代码走查和技术评审。其中代码走查和代码审查是由若干个程序员和测试人员组成的一个小组，集体讨论。这两个方法都需要先做一些准备工作，然后才举行会议进行讨论，会议的主题是找出软件的问题——不仅是出错的项目，还包括遗漏的项目，但不解决问题。很多软件项目团队选择审查作为评审核心代码的方式，将走查和同级桌查作为一般代码的评审方式。

1. 代码审查

代码审查的内容有：代码和设计的一致性、代码执行标准的情况、代码的逻辑表达正确性、代码结构的合理性、代码的可读性等。代码审查是在开发组内部进行的，类似于“如果你给我看你的，我也给你看我的”。

代码审查一般不少于 4 人，分别为组长、资深程序员、程序编写者与专职测试人员。组长不能是被测程序的编写者，组长负责分配资料、安排计划、主持开会、记录并保存发现的差错。

代码审查组采用讲解、提问并使用检查表的方式审查代码，寻找问题和失误。一般有正式的计划、流程和结果报告。同时为了保证审查的效率，所有的参与者要保证正式审查的 4 个关键要素：查找问题、遵守规则、审查准备和编写报告。

代码审查由 4 个步骤组成：准备、程序阅读、审查会、编写报告。

(1) 准备：将程序目录表和设计说明书等材料交给小组成员，要求大家熟悉这些材料，并由设计人员和编程人员对所提交的材料进行说明，以了解代码的主要功能和各个功能之间的联系。

(2) 程序阅读：审查组人员在对程序有了一定的了解后，仔细阅读代码和相关的材料，标出明显的缺陷及错误。

(3) 审查会：由程序员阐明程序的逻辑，其他人员提出问题，确定错误是否存在，然后采用代码审查会来分析讨论，切记是发现问题，不是解决问题。

(4) 编写报告：在会后审查人员除了要编写审查结果外，还要将发现的错误编写成报告交给程序开发人员，其中错误报告也要分析、归类和提炼。审查会议的结果必须尽快告诉别人——例如发现了多少错误，在哪里发现的。

代码审查过程中最主要的是代码审查清单，这里给出一种常用的代码审查清单：

(1) 数据引用错误。数据引用错误主要有：①是否引用未初始化的变量？查找遗漏之处和查找错误同样重要；②数组和字符串的下标是整数值吗？下标是否总是在数组和字符串的长度范围之内？③用下标引用数组时是否存在“差 1 错误”？④在引用指针或变量时是否已分配内存？

(2) 数据声明错误。数据声明错误主要有：①所有变量是否都被赋予正确的长度、类型和存储类？②是否存在已声明但从未用过的变量？③所有变量是否都显式声明了？

(3) 计算错误。计算错误主要有：①计算中是否使用了不同数据类型的变量？②计算中是否存在混合运算？③计算中是否出现了溢出现象？④对于整型运算，处理某些计算是否存在精度丢失问题？⑤除数/模是否为零？⑥赋值语句的目标变量是否比含有变量的表达式的取值范围小？⑦是否考虑和了解到编译器对类型和长度不一致的变量的转换规则？等等。

(4) 子程序参数错误。子程序参数错误主要有：①子程序接受的参数类型与调用代码发送的匹配吗？②常数是否当作形参传递，在子程序中被改动？③参数类型是否与相应的形参匹配？④在子程序中是否定义了与全局变量相同的变量？

(5) 静态结构问题。静态结构问题主要有：①函数的调用关系是否正确；②是否存在孤立的函数而没有被调用；③编码的规范性；④资源是否释放；⑤数据结构是否完整和正确；⑥是否有死代码和死循环；⑦代码本身是否存在明显的效率和性能问题；⑧代码本身的方法、类和函数的划分是否清晰、易理解；⑨代码本身是否健壮，是否有完善的异常处理和错误处理。

在这里只是介绍了一部分，其他的还有控制流错误、比较错误、输入/输出错误，等等。

2. 桌面检查

桌面检查就是程序员自己检查自己所编写的程序，即对源程序代码进行分析、检验，根据相关的文档，检验程序中是否存在错误的过程。

桌面检查主要做的工作是：①检查代码和设计是否一致；②代码是否遵循标准、是否可读；③代码逻辑表达是否正确；④代码结构是否合理；⑤程序编写与编写标准是否符合；⑥程序中是否有不安全、不明确和模糊的部分；⑦编程风格是否符合要求。

桌面检查更加细致的工作是：①检查变量的交叉引用表(是否有未说明的变量和违反了类型规定的变量)；②检查标号的交叉引用表(验证所有标号是否正确)；③检查子程序、宏、函数(验证每次调用与所调用位置是否正确，调用的子程序、宏、函数是否存在，参数是否一致)；④检查全部等价变量的类型的一致性；⑤确认常量的取值和数制、数据类型；⑥选择、激活路径(在设计控制流图中选择某条路径，到实际的程序中激活这条路径，如果不能激活，程序可能有错)；⑦对照程序的规格说明，详细阅读源代码，比较实际的代码，从差异中发现程序中的问题和错误。

桌面检查是一种很老的方法，但效率不高，这是因为：①人都有思维定式，有些习惯性错误自己很难发现；②人们对于自己的程序都有一种偏爱心理，没有发现错误的欲望；③如果是对功能的理解错误，则自己很难纠正。因此这种方法只能用于个人自我检查和发现一些较明显的错误和漏洞。

3. 代码走查

代码走查和代码审查类似，代码审查是一种正式的评审活动，而代码走查的讨论过程是非正式的。当然走查比审查要更具技术性一些。在代码走查中编写代码的程序员要向 5 人小组或由其他程序员和测试人员组成的小组做正式陈述。在这个小组中至少有一位资深程序员，测试人员应是具有经验的程序设计人员或精通程序设计的人员，还要有一位没有介入这个项目的新人(这样的人不会被已有的设计约束，以发现问题)。

代码走查的过程和代码审查相似，先把材料交给审查者，审查者阅读材料发现错误，在审查会期间由陈述者(编写代码的程序员)逐行或逐段地通读代码，解释代码为什么以及如何工作。审查人员聆听叙述、提出有疑义的问题，最后审查小组做出审查结果的书面报告和错误报告，交给程序开发人员。

代码走查主要有文档和源程序代码、检查功能、检查界面、检查流程、检查提示信息、函数检查、数据类型与变量检查、条件判断检查、循环检查、输入/输出检查、注释检查、程序(模块)检查、数据库检查、表达式分析、接口分析、函数调用关系图及模块控制流图17项检查内容。

(1) 要求有文档和源程序代码(一份最新的设计文档、程序结构图、所有模块的源程序代码、代码体系结构描述、目录文件、代码组织等)。

(2) 检查功能(重复的功能，多余的功能，功能实现与设计要求不相符，功能的可使用性、方便性和可用性)。

(3) 检查界面(界面美观，控件排列、格式，焦点控制的合理或全面性)。

(4) 检查流程(流程控制是否符合要求，流程实现是否完整)。

(5) 检查提示信息(提示信息重复或出现时机的合理性，提示信息格式和要求的合理性，提示框返回后停留位置的合理性)。

(6) 函数检查(函数声明部分清楚地描述函数及其功能，代码中有相关注解，函数名清晰地定义了它的目标及功能，函数只做一件事情，函数的参数都被使用，函数的参数接口关系清晰，函数出口都有返回值，函数异常处理清楚)。

(7) 数据类型与变量检查(数据有效性检测是否合理，数据来源正确性，数据处理过程正确性，数据处理结果正确性，数据类型解释，变量分配了正确的长度、类型和存储空间，静态变量明确区分，变量初始化，变量的命名不与标准库中的命名相冲突，全局变量描述，类型转换)。

(8) 条件判断检查(if/else使用正确，无嵌套的if链，数字、字符、指针和0/NULL/FALSE判断明确，不要有臃肿的判断逻辑，所有的判断条件边界是否正确，判断体足够短)。

(9) 循环检查(循环体不为空，循环之前做好初始化代码，有明确的多次循环操作可使用for循环，不明确的多次循环操作可使用while循环，循环终止的条件清晰，所有的循环边界是否正确，循环体内的循环变量起到指示作用)。

(10) 输入/输出检查(所有文件的属性描述清楚，输入参数的异常是否处理，对文件结束的条件进行检查)。

(11) 注释检查(有简单的说明用于描述代码的结构，每个文件和模块均已给予解释，解释说明代码功能且准确描述代码意义，解释不要过于简单，注解清楚正确，代码的注释与代码是否一致且注释是否多余)。

(12) 程序(模块)检查(程序中所有的异常是否处理；程序中是否存在重复的代码，程序结构是否清晰)。

(13) 数据库检查(数据库命名使用小写英语字母、数字和下画线(无其他字符)，数据库命名采用项目名称或产品名称命名(长度小于20位)，数据库中的所有表字符集统一，数据库对象的命名不使用保留关键字，数据库设计考虑到将来可能存在的异种数据库迁移，字段与画面项目能够一一对应(部分标识符字段和系统设定字段除外)，索引是多值字段，索

引是单一字段，字段取值符合域定义，字段的类型和长度能够满足字段值的最大限量，文本字段有充足的余量对应可能的长度变更，数字字段考虑了充足的余量和精度对应可能的长度或精度变更，针对客户的特定应用采用了视图机制)。

(14) 表达式分析(对表达式进行分析以发现和纠正表达式中出现的错误，如在表达式中不正确地使用了括号而造成错误、数组下标越界错误、除数为零、浮点数计算的误差等)。

(15) 接口分析(主要是对接口一致性的分析，如各模块之间接口一致性，模块与外部数据库的接口一致性，形参与实参在类型数量、顺序、维数、使用上的一致性，全局变量和公共数据区在使用上的一致性)。

(16) 函数调用关系图(通过应用程序各函数之间的调用关系展示系统的结构。方法是列出所有函数，用连线表示调用关系。作用是可以检查函数的调用关系是否正确，是否存在孤立的函数而没有被调用，明确函数被调用的频繁度，对调用频繁的函数可以重点检查)。

(17) 模块控制流图(模块控制流图是由许多节点和连接节点的边组成的图形，其中每个节点代表一条或多条语句，边表示控制流向，可以直观地反映出函数的内部结构)。

4. 技术评审

技术评审是最正式的审查类型，具有高度的组织化，要求每一个参与者都接受训练。技术评审由开发组、测试组和相关人员(QA、产品经理等)联合进行，综合运用走查、审查技术，逐行、逐段地检查软件。技术评审与走查和审查的不同之处在于表述代码的人——表述者，表述者不是原来编写代码的程序员，这就迫使表述者学习和了解要表述的材料，从而有可能对程序提出不同的看法和解释。检查的要点是设计需求、代码标准/规范/风格和文档的完整性与一致性。

经验表明，通过代码审查、桌面检查、代码走查、审查和技术评审能够有效地发现30%~70%的逻辑设计和编码错误，而且这种方法一次能解释一批错误，同时还能对错误进行定位。

7.2.2　代码编程规范检查

编程规范又称为代码规则、编码规则，是对程序代码的格式、注释、标识符命名、语句使用、函数、类、程序组织、公共变量等方面所做的要求。如果软件都能在编写的阶段遵循一定的编程规范，这对软件产品的质量将大有好处：遵循一定的编程规范，使得开发人员书写的代码更健壮、更安全、更可靠，可以提高代码的可读性、可重用性、可移植性和可维护性，使代码易于查看和理解，是提高代码质量最有效、最直接的手段。

不少公司大力推行 CMM(软件能力成熟度模型)或 TSP(团队软件过程)，也有不少公司对程序员进行 PSP(个人软件过程)推行，无论推行什么开发过程，目的都只有一个，就是要求规范统一，以便整个开发团队便于交流、步调一致。遵守编程规范是一名合格程序员的必要条件，也是从学生程序员蜕变为职业程序员的重要标志。在公司团队协作开发的情况下，编程时应该强调的一个重要方面是程序的易读性，在保证软件的速度等性能指标满足用户需求的情况下，能让其他程序员容易读懂你的程序。一套鲜明的编程风格，可以让协作者、后继者和自己一目了然，在很短的时间内看清程序的结构，理解设计的思路，从而极大有效地促进程序员之间的交流。

然而对于各个不同的领域、行业、语言、操作系统、版本、公司、团队甚至不同的项目，都需要符合自己特点的编程规范，编程规范的种类可以说是五花八门、数不胜数。虽然有一些由业内比较著名的组织或公司指定的相对权威的编程规范，但由于矛盾的特殊性，这些权威的编程规范更多的是被参考和引用，而并不能被应用到所有的应用领域。比如，编译程序和解释性程序的编程规范是明显不同的，基于 Web 的语言更有它独特的要求，有时如果强制性地遵循某些权威或比较流行的规范，可能会导致效率下降或可读性不强等问题。

下面具体讲述一下编码规则的具体内容。

1. 对代码书写格式的要求

我们先比较一下下面两段代码：

程序 A：

```
int main()
{
  int i，b，c，a[]={...}，*p，*q；
  b=c=1；p=q=a；
  for(i=0；i<6；i++)
  {if(b<*(a+i)){b=*(a+i)；p=&a[i]；}if(c>*(a+i)){c=*(a+i)q=&a[i]j；}}
  i=*a；*a=*p；*p=i；i=*(a+5)；*(a+5)=*q；*q=i；
  printf("%d，%d，%d，%d，%d，%d\n"，a[0]，a[1]，a[2]，a[3]，a[4]，a[5])；
}
```

程序 B：

```
int main()
{
    int i，b，c，a[]={...}，*p，*q；

    b=c=1；
    p=q=a；

    for(i=0；i<6；i++)
    {
        if(b<*(a+i))
        {
            b=*(a+i)；
            p=&a[i]；
        }

        if(c>*(a+i))
        {
            c=*(a+i)
            q=&a[i]；
        }
    }

    i=*a；
    *a=*p；
    *p=i；
```

```
        i=*(a+5);
        *(a+5)=*q;
        *q=i;

        printf("%d，%d，%d，%d，%d，%d\n"，a[0]，a[1]，a[2]，a[3]，a[4]，a[5]);
    }
```

实现同样功能的两段程序，都可以编译通过，正确运行。但是两者相比较而言，程序 B 明显要比程序 A 的可读性好，程序 B 可以很清楚地看到程序的模块，查找错误时也非常方便，而程序 A 虽然可以编译通过，正确运行，但如果想做什么修改或维护，则很难清楚明白整个程序，无从下手；另外，程序 B 比程序 A 占用的空间大，但是在程序编译运行中，空格和空行是没有影响的。下面详细介绍代码书写格式。

1) 空行、空格的使用

由于在程序的编译和运行过程中，空行和空格没有影响，因此可以利用空行和空格使程序看起来整齐、清楚。如同上面的程序 B 一样，利用空行和空格将程序在哪一步做了什么事情划分得清清楚楚，使人一目了然。

2) 对程序语句要按其逻辑水平缩进

缩进可以让人清楚地看到程序的逻辑结构，有助于对程序进行维护。例如在程序 B 中，可以清楚地看到 for 循环语句的作用范围，也可以很清楚地检查到程序中存在的逻辑错误。尤其是在很复杂的程序中，存在很多的嵌套循环，可以利用缩进清楚明白地看到程序块的逻辑结构。

3) 一行只写一条语句，一次只声明、定义一个变量

在程序 A 中，一行写了很多条语句，使人看起来眼花缭乱，在查找错误时，不易查找；在程序 B 中，每一行只写一条语句，再加上空行和空格的使用，使程序看起来简洁清楚，易于查错和维护。

在表达式中使用括号：

表达式 1：a>b&&x>y

表达式 2：(a>b)&&(x>y)

表达式 1 和表达式 2 表示的是同样的意思，但是表达式 2 看起来就很清楚，不会让人产生歧义，而表达式 1 则会产生歧义。由于运算符有优先级，不同运算符的优先级不同，运行的先后自然也不一样，但是当表达式中有众多运算符时，最好加上括号以表明运算符运行的优先级，这样方便查找程序中存在的逻辑错误，利于维护。

2. 程序中的注释

程序中的注释是程序与日后程序读者之间通信的重要手段，良好的注释能够帮助读者理解程序，为后续阶段进行测试和维护提供明确的指导。

注释的基本原则：①注释内容要清晰明了，含义准确，防止出现二义性；②边写代码边写注释，修改代码的同时也要修改相应的注释，以保证代码与注释的一致性。

通常需要添加注释的内容有函数、类、文件、空循环体，对 switch 语句中多条 case

语句共用一个出口的情况及其他进行注释，在行末注释时尽量对齐注释。在程序中注释行的数量不得少于程序行数量的 1/3。

3. 命名

不同的编程语言对于变量、文件名、常量、函数的命名都有各自统一的命名规范。例如在 C 语言中，用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列统称为标识符，标识符只能由字母、数字和下画线组成，而第一个字符必须为字母或下画线。例如，下面列出的就是合法的标识符：

```
Sum， li_ling， class1， _day3
```

下面的是不合法的标识符：

```
M.D.， $123， 34th， a>b
```

另外，在 C 语言中还要求标识符的长度不能超过 32 个字符，还要尽量做到“见名知义”，即选择有含义的英文单词或缩写作为标识符，如 count、name、day、month 等，符号常量名用大写，变量名用小写。文件名的命名类似。

4. 语句

对于具体程序语句的使用要求，不同的编程语言有不同的要求，在这里简单介绍几点：

(1) 禁用 goto 语句，goto 语句使程序的流程无规律、可读性差。

(2) new和delete要成对出现，new用来分配一块新的内存，delete用来释放一块内存。如果二者不成对使用，将会造成内存的浪费。

(3) 对 switch 语句中的每个分支都要以 break 语句结尾。

(4) 指针要初始化，释放内存后的指针变量要赋 NULL 值。

5. 函数

在声明和定义函数时，要在函数参数列表中为各个参数指定类型和名称；为每一个函数指定其返回值，若没有返回值，则要定义返回类型为 void；若函数体代码的长度超过 100 行(不包括注释行)，则需要重写编写函数；另外在函数中要注意全局变量的使用。

6. 类

类的成员变量的声明必须写在该模块中所有可执行代码之前；应当至少声明一个构造函数，而且类的构造函数应当出现在所有成员函数的最前面，应当以参数个数递增的顺序排列；类的成员变量和成员函数的出现应当根据其可访问性的级别；在派生类中不要对基类中的非虚函数重新进行定义，如果确实需要在派生类中对该函数进行不同的定义，那么应在基类中将该函数声明为虚函数；用内联函数代替宏函数；若重载了操作符 new，则也要重载 delete；虽然类可以继承，但是仍要限制类的继承层数(最好不要超过 5 层)。

7. 程序组织

由于程序的规模越来越大，软件往往包含多个文件，因此在程序的组织中要注意：①一个头文件中只声明一个类；②一个源文件中只实现一个类；③头文件中只包含声明，不

应包含定义或实现；④源文件中不要有类的声明；⑤只允许头文件被包含到其他代码文件中；⑥避免头文件重复包含。

有了统一的规范后，测试工程师或程序员自身就可以实施编码规范检查了。要真正把编码规范贯彻下去，单单靠测试员、程序员的热情，很难坚持下去，我们必须借助一些专业的工具来实施。在 C/C++语言的编程规则检查方面，比较专业的工具有 C++ Test、LINT、QAC/QAC++等，这些工具通常可以和比较流行的开发工具集成在一起，程序员在编码过程中，在编译代码的同时便完成了编程规则的检查。

7.2.3　代码的自动分析

代码的自动分析需要用到代码分析工具，代码自动分析的结果可以对照需求和设计文档以及编码进行检查，主要进行程序逻辑和编码检查、一致性检查、接口分析、I/O 规格说明分析，以及数据流、变量类型检查和模块分析等。代码自动分析的结果可以作为动态测试和其他测试的必要准备。

运用代码分析工具进行代码自动分析的内容主要是：生成引用表、进行程序错误分析和接口分析。

1. 生成引用表

代码分析所生成的引用表有：循环层次表、变量交叉引用表、标号交叉引用表、子程序引用表、等价表、常数表、操作符统计表和操作数统计表。

生成引用表的目的：直接从表中查出说明、使用错误，例如循环层次表、变量交叉引用表、标号交叉引用表；为用户提供辅助信息，例如子程序引用表、等价表、常数表；用来做错误预测和程序复杂度的计算，例如操作符和操作数的统计表。

1) 标号交叉引用表

列出各模块出现的全部标号(可以是按标号出现的先后顺序，也可以按字典顺序)；在表中标出标号的属性——已说明、未说明、已使用、未使用；在表中还可以包括模块外的全局标号、计算标号。

2) 变量交叉引用表

变量交叉引用表也称变量定义与引用表(在表中，变量的顺序可以是按变量出现的先后顺序，也可以按字典顺序，还可以按它们的类型排序)；表中应标明各个变量的属性——已说明、未说明、私有/公有说明，以及类型和使用情况。

3) 子程序、宏和函数表

在表中列出各个子程序、宏和函数的属性，包括已定义、未定义、定义类型；已引用、未应用、引用次数；输入参数的个数、类型、顺序；输出参数的个数、类型、顺序。

4) 等价表

在表中列出在等价语句或等值语句中出现的全部变量和标号。

5) 常数表

在表中列出全部的数字常数和字符常数，并指出他们在哪些语句中首先被定义。

2. 程序错误分析

程序错误分析的目的是用于确定在源程序中是否有某类错误或危险结构。分析的内容有：变量类型和单位分析、引用分析以及表达式分析。

1) 变量类型和单位分析

即为了强化对源程序中数据的检查，在程序设计语言中扩充一些新的数据类型。例如，仅能在数组中使用的下标类型以及在循环语句中当作控制变量使用的计数器类型。这样就可以应用静态预处理程序，分析程序中的类型错误。

2) 引用分析

在静态错误分析中，最广泛使用的技术就是发现引用异常。例如，我们沿着程序的控制路径，变量在赋值以后未被引用，这就发生了引用异常。为此，我们需要检查通过程序的每一条路经。可以用类似深度优先算法的方法来遍历程序流程中的每一条路径；建立引用异常的探测工具。这种工具包括两个表：定义表和引用表。每张表中都包含一组变量名。未引用的表中包括已被赋值，但未被引用的一些变量。

3) 表达式分析

对表达式进行分析，可以发现和纠正在表达式中出现的错误。在表达式中可能存在：不正确使用括号造成的错误，数组下标越界造成的错误，除数为零造成的错误，对负数开平方造成的错误。其中最复杂的一类表达式分析是对浮点数计算的误差进行检查。

3. 接口一致性分析

接口一致性分析的目的是检查模块之间接口的一致性和模块与外部数据库之间接口的一致性。

接口一致性分析的内容：

(1) 检查形参与实参在类型、数量、维数、顺序、使用上的一致性。

(2) 检查全局变量和公共数据区在使用上的一致性。

7.2.4　代码结构分析

代码的结构形式是白盒测试的主要依据。研究表明，程序员 38%的时间花费在理解软件系统上，因为代码以文本格式被写入多重文件中，这是很难阅读理解的，需要其他一些东西来帮助人们阅读理解，如各种图表等，而代码结构分析满足了这样的需求。

在代码结构分析中，测试者通过使用测试工具分析程序源代码的系统结构、数据结构、内部控制逻辑等内部结构，生成函数调用关系图、模块控制流图、模块数据流图、内部文件调用关系图、子程序表、宏和函数参数表等各类图形图表，可以清晰地标识整个软件系统的组成结构，使其便于阅读和理解，然后可以通过分析这些图表，检查软件是否存在缺陷或错误。

1. 函数调用关系图

函数调用关系图/程序调用关系图都是对源程序中函数关系的一种静态描述，即通过应用程序中各函数之间的调用关系展示系统的结构。在函数调用关系图中，节点表示函数，

边表示函数之间的调用关系。

通过查看函数调用关系图，可以检查函数之间的调用关系是否符合要求，是否存在递归调用，函数的调用是否过深，是否存在独立的没有被调用的函数。从而可以发现系统是否存在结构缺陷，发现哪些函数是重要的，哪些是次要的，需要使用什么级别的覆盖要求，等等。图 7-3 所示为函数调用关系图。

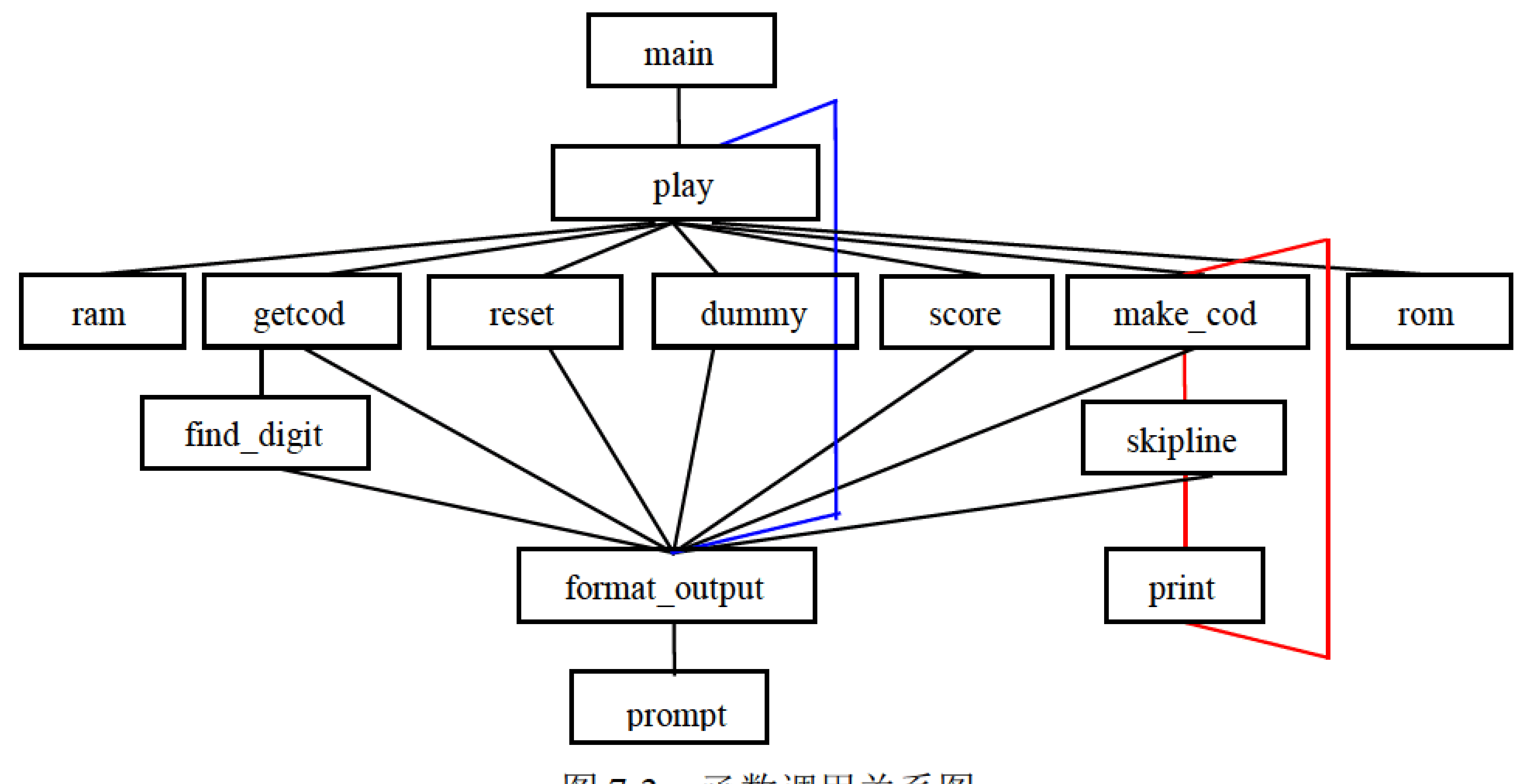


图 7-3　函数调用关系图

图 7-3 指出了程序中的很多问题：程序的层次性较差、存在直接和间接的递归调用以及重要资源被诸多模块使用等。

函数调用关系图在软件工作领域有广泛的应用，如编译优化、过程间数据流分析、回归测试、程序理解等。

2. 模块控制流图

模块控制流图是与程序流程图类似的由许多节点和连接节点的边组成的一种图形，其中一个节点代表一条语句或数条语句，边代表节点间控制流向，它显示了一个函数的内部逻辑结构。模块控制流图可以直观地反映一个函数的内部逻辑结构，通过检查这些模块控制流图，能够很快发现软件中的错误与缺陷。

通过控制流图，可以很直观地发现程序中的问题，例如在图 7-4 中，我们发现程序中使用了 goto 语句、代码重复、开关语句结构有问题以及死代码等。

因此，我们对程序结构提出 4 点基本要求，这些要求是，写出的程序不应该包含：①转向并不存在的标号；②没有用过的语句标号；③从程序入口进入后无法到达的语句；④不能达到停机语句的语句。

3. 模块数据流图

在单元测试中，数据仅仅在一个模块或函数中流通。但是，数据流的通路往往涉及多个集成模块，甚至于整个软件，所以我们有必要进行数据流的分析，尽管它非常耗时。

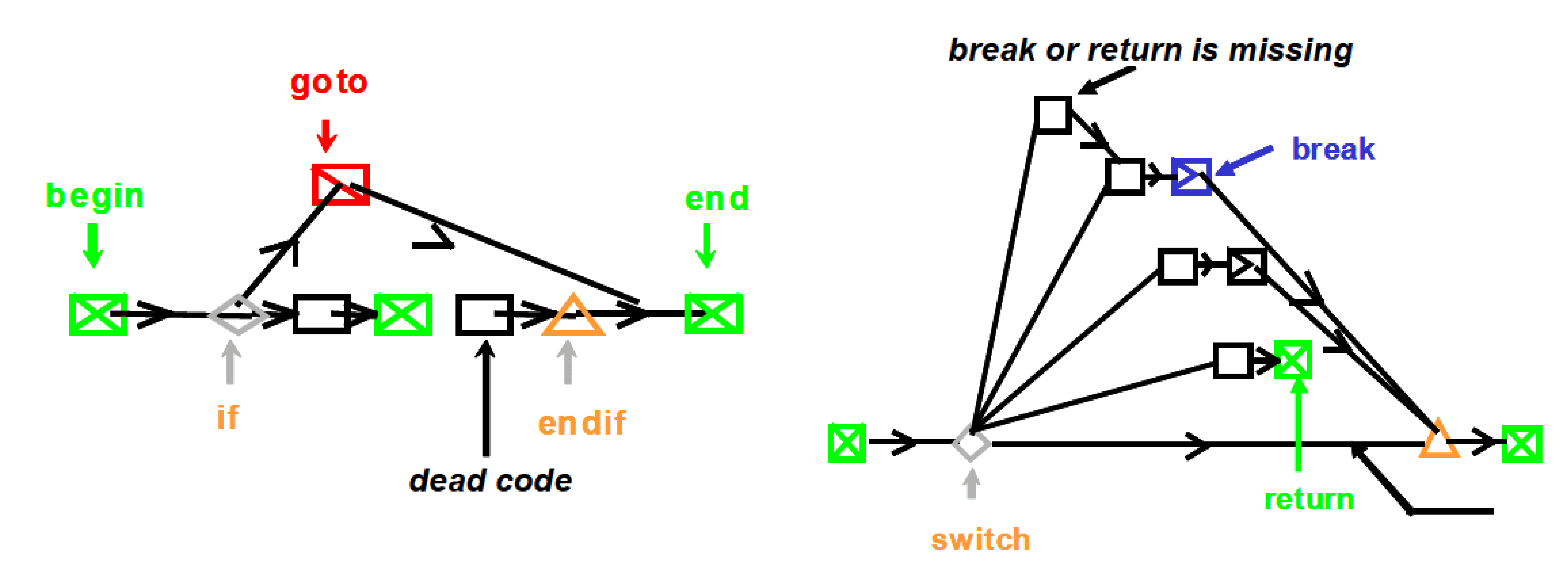


图 7-4　模块控制流图

数据流分析技术最早被用于编译优化，目前除编译优化以外，在程序测试、程序理解、程序验证、程序调试以及程序分片等许多领域，数据流都有着广泛应用。特别是近年来数据流分析方法在确认系统中也得到成功应用，用以查找诸如引用未定义变量等程序错误。也可以用来查找对以前未曾使用的变量再次赋值等数据流异常情况。找出这些错误是很重要的，因为这常常是常见程序错误的表现形式，如错拼名字、名字混淆或是丢失了语句。这里将首先说明数据流分析的原理，然后指明它可揭示的程序错误。

在程序中，如果某条语句执行时能改变某程序变量 V 的值，则称 V 是被该语句定义的。如果某条语句的执行引用了内存中变量 V 的值，则称该语句引用变量 V。例如，语句：

```
X := Y + Z
```

定义了 X，引用了 Y 和 Z，而语句：

```
If Y > Z then   goto exit
```

只是引用了 Y 和 Z。

输入语句：

```
READ X
```

定义了 X。

输出语句：

```
WRITE X
```

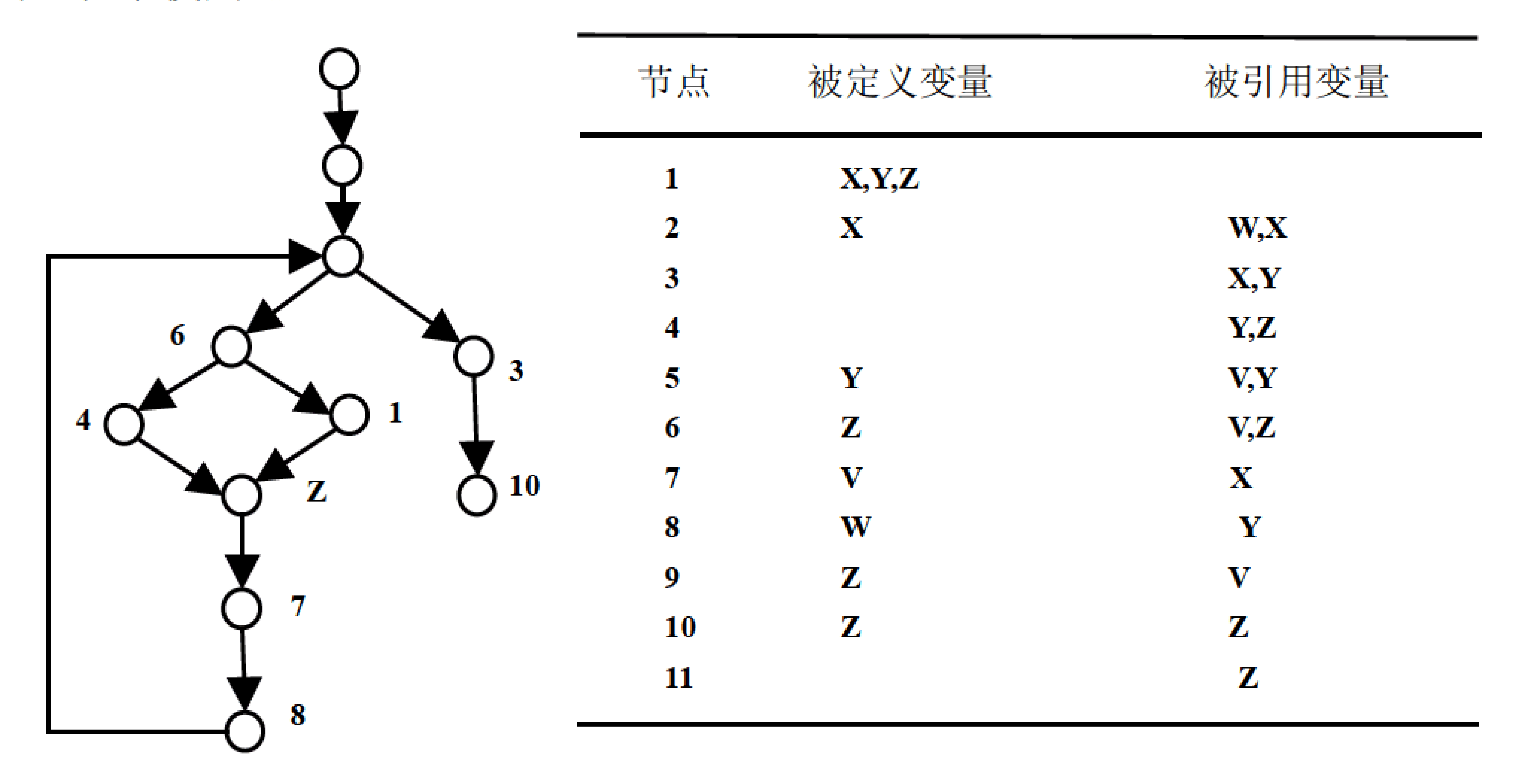
引用了X。执行某条语句也可能使变量失去定义，成为无意义的语句。例如，在FORTRAN中，循环语句DO的控制变量在经循环的正常出口离开循环时，就变成无意义的变量。

图 7-5 所示为一个小程序的控制流图，同时指明了每一条语句定义和引用的变量。可以看出，第一条语句定义了 3 个变量 X、Y 和 Z，这表明它们的值在程序外部被赋予。例如，该程序是以此三变量为输入参数的过程或子程序。同样，出口语句引用 Z 表明：Z 的值被送给外部环境。

该程序中含有两个错误：①语句 2 使用了变量 W，而在此之前并未对其定义；②语句 5 和 6 使用了变量 V，这在第一次执行循环时也未对其定义。

此外，该程序还包含两个异常：①语句 6 对 Z 的定义从未使用过；②语句 8 对 W 的

定义也从未使用过。



节点	被定义变量	被引用变量
1	X,Y,Z	
2	X	W,X
3		X,Y
4		Y,Z
5	Y	V,Y
6	Z	V,Z
7	V	X
8	W	Y
9	Z	V
10	Z	Z
11		Z

图 7-5　控制流图及其定义和引用的变量

当然，程序中包含一些异常，语句 3 和 4 还会引起执行错误。不过这一情况表明，也许程序中含有错误；也许可以把程序写得更容易理解，从而能够简化验证工作以及随后的维护工作(去掉那些多余的语句一般会缩短执行时间，不过在此我们并不关心这些)。

总之，在数据流最初分析中，我们可以集中关注定义/引用异常的缺陷，例如：变量被定义，但是从来没有使用过；所使用的变量没有被定义；变量在使用之前被定义了两次。另外，因为程序中的语句因变量的定义和使用而彼此相关，所以用数据流测试方法更能有效地发现软件缺陷。但是，在度量测试覆盖率和选择测试路径的时候，数据流测试很困难。

对数据流进行分析的工作可借助编译器或程序分析工具来完成。

7.2.5　代码安全性检查

所谓代码安全性，就是代码在运行时或被别人调用时产生错误的容易程度。如 C++，它规定了严格的语法，然而又有其灵活性，这种灵活性增加了程序的不可预见性，这直接导致故障的发生，致使代码的安全性变差。例如指针的指向发生错误，则直接导致结果错误；另外指针的指向越界，可能导致缓冲区溢出，很多病毒就是利用缓冲区溢出对计算机进行攻击的；还有 C++提供的一些关于字符处理的函数，虽然提供了这方面的功能，但没有对参数范围进行限制和检查，这也很容易导致错误的发生。

代码安全性检查或静态错误分析主要用于确定在源程序中是否有某类错误或危险/不安全的结构。一般可借助工具来对代码的安全性进行检查。

1. 代码安全性检查方法

对代码进行安全性检查大致有四种方法：

1) 对变量类型和单位进行检查

为了强化对源程序中数据类型的检查，发现数据类型上的错误和单位上的不一致性，在程序设计语言中扩充了一些结构。比如单位分析要求使用一种预处理器，它能够通过使

用一般的组合/消去规则，确定表达式的单位。

2) 对变量引用进行检查

最常用的代码安全性检查方法就是发现引用异常。如果沿着程序的控制路径，变量在赋值以前被引用，或变量在赋值以后未被引用，这时就发生了引用异常。为了检测引用异常，需要检查通过程序的每一条路径。也可以建立引用异常的探测工具。

3) 对表达式运算进行检查

对表达式进行检查，以发现和纠正表达式中出现的错误。包括：在表达式中因不正确地使用括号造成错误，数组下标越界造成错误，除数为零造成错误，对负数开平方或对 π 求正切值造成错误，以及对浮点数计算的误差进行检查。

4) 接口分析

对接口进行安全性检查主要检查模块、过程、函数以及它们之间接口的一致性。因此要检查形参与实参在类型、数量、维数、顺序、使用上的一致性；检查全局变量和公共数据区在使用上的一致性。

2. 代码安全性检查关注的错误

具体而言，代码安全性检查要发现如下错误：

1) 坏的存储分配

内存的申请和释放都有相应的函数，二者必须一一对应才能将申请的内存释放掉。用 maloc/caloc/realloc 申请的内存，对应用 free 来释放；用 new 申请的内存，必须用 delete 来释放。如果对应不匹配，就会发生错误。

2) 内存泄漏

程序在执行过程中，在内存中动态申请的内存在函数返回或程序退出时必须有相应的释放操作，没有执行释放操作，将导致内存丢失，尤其在循环体中，变量的循环累积有可能造成系统崩溃。

4) 指针引用

指针变量在使用前必须确保指向确定的地址单元，如果指针为空或指向错误的地方，就会导致程序出现异常。

5) 约束检查

越界指针、数组越界就是指针或数组超出了原先设定的范围，导致出现意想不到的结果或异常。

6) 变量未初始化

变量定义后必须被初始化，未初始化的变量的值不确定，使用它会使程序出现不正确的结果，甚至导致程序出现异常。

7) 错误逻辑结构

检查在逻辑上可能有错误的结构以及多余的不可达的程序段(不可达代码是指永远执行不到的代码)，如交叉转入/转出的循环语句、为循环控制变量赋值、存取其他模块的局

部数据等。

8) 其他

缓冲区溢出、非法类型转换、非法的算数运算(例如，被零除错误、负数开方)、整数和浮点数的上溢出/下溢出、多线程对未保护数据的访问冲突等都可能导致出现异常。

7.3　软件复杂性分析

软件危机产生的最直接原因是软件复杂性已远远超出人们对复杂性控制的能力。软件复杂性越高，软件中隐含错误的概率越大，软件的可靠性、可维护性越差。近几十年来的研究表明，作为软件显著特点的复杂性是导致软件错误的主要原因，软件可靠性问题的本质就是复杂性问题。软件的可靠性与其复杂性密不可分，当复杂性超过一定限度时，软件缺陷或错误便急剧上升，甚至引发软件开发失败。此外，软件的可维护性等质量特性也与之有极大关系。1979 年，Belady 和 Lehman 提出的软件维护模型表明，软件的维护开销除了与维护人员的素质等相关外，维护工作量是复杂性的一个指数函数。由此可见，软件复杂性度量与控制是软件可靠性工程亟待解决的一个重要问题。而对软件复杂性进行分析、度量和控制，其目的就是减少由软件设计方法和技巧使用不当而带来的复杂性，可以更好地对软件开发过程进行控制，并降低由复杂性引发软件错误的可能性，提高软件的可靠性和可维护性，确保软件产品的质量。

对软件复杂性的研究已经有几十年的历史，人们对软件复杂性的基本认识是：越复杂的事物越容易出错，并带来问题。软件复杂性反映分析、设计、测试、维护和修改软件的困难程度或复杂程度，而且这种复杂性将来会与日俱增。

软件复杂性产生的原因主要有如下几个方面：

(1) 软件应用需求太复杂、应用要求太高。

(2) 软件开发环境(包括编程、调试、测试、应用仿真等)及应用环境太复杂。

(3) 软件应用框架、结构及模型太复杂。

(4) 软件开发过程和开发模型太复杂。

(5) 软件项目因涉及人的智力劳动，导致管理太复杂。

(6) 软件设计与验证太复杂，尤其是嵌入式应用软件的设计与验证。

总之，软件复杂性是在软件开发过程中逐渐产生的，最终主要体现在软件结构复杂性和算法复杂性等方面。

7.3.1　软件复杂性度量与控制

软件复杂性度量是对软件复杂性的定量描述，是软件复杂性分析和控制的基础。软件复杂性度量的结果是软件复杂度。对象不同，描述软件复杂性的角度和方法不同。程序长度等一些经典方法曾发挥过积极的作用，但它们仅仅反映了软件规模，没有真实地反映软件的复杂性。于是，人们又根据软件结构，从数据流和控制流角度出发，结合软件的模块复杂性、结构复杂性以及这两者的总体复杂性度量，来实现对软件复杂性的度量。这是一

种从本质上反映软件复杂性的综合方法。不过，基于总体和全局的，能从本质上综合反映软件复杂性的精确度量方法尚待进一步研究。

目前，人们已经研究出来许多度量方法和标准，但主要分为两大类：面向对象的软件复杂性度量和面向过程的软件复杂性度量。而研究最为活跃、成果也较多的是面向过程的软件复杂性度量。其中著名的软件复杂性度量方法有 Line Count 语句行度量、基于 FPA 功能点分析的度量、Halstead 软件科学度量法和 McCabe 结构复杂性度量。随着近几年面向对象技术的兴起，面向对象分析(OOA)、面向对象设计(OOD)、面向对象程序设计(OOP)的技术、方法和工具发展很快，且得到广泛应用。面向对象复杂性度量方法 C&K、MOOD 等的提出和应用标志着软件复杂性度量进入面向对象度量的阶段。C&K 度量方法主要考虑类的继承、类的方法数、类之间的耦合、类的内聚性等，并在此基础上提出一整套面向对象度量方法。MOOD 度量方法是从封装、继承、耦合、多态性提出类的多个度量指标，提供对一个系统的基本判断，对软件工程管理者有一定的用处。

虽然上述软件复杂性度量方法都有各自的缺点和不足，但在一定程度上，从不同的侧面反映了软件的复杂性，其中许多方法已被成功地运用到软件开发中。

事实上，即便是最精确的度量，软件复杂性度量也只是为软件复杂性的定量分析和控制提供依据，其根本目的是通过控制软件的复杂性来改善和提高软件的可靠性。设计过程是软件复杂性形成的根源，因此最有效的办法是在软件设计过程中对软件复杂性进行有效控制，使之保持在一个合理的范围内。

1. 软件复杂性度量

目前软件复杂性度量主要根据软件结构，从数据流和控制流角度出发，结合软件的模块复杂性、结构复杂性以及这两者的总体复杂性度量，来实现对软件复杂性的度量。这是一种从本质上反映软件复杂性的综合方法。

1) 模块复杂性度量

模块复杂性度量是软件复杂性度量的基础，是一种曾一度被广泛使用的传统方法，主要有 McCabe 度量和程序长度(Halstead)度量两种方法。软件结构复杂性包含模块结构的复杂性和整个软件结构的复杂性两个方面。

2) 模块结构复杂性度量

模块结构复杂性度量主要用来对软件中的数据流和控制流结构、模块信息流结构和模块之间互连的复杂程度等进行度量和评价。

模块结构复杂性度量力图反映模块内部结构复杂性、模块之间的调用关系(即接口复杂性)，以及常用模块的扇入/扇出数量或信息的扇入/扇出数量(模块的扇入等于进入该模块的信息流与该模块的输入数据之和，模块的扇出等于进入该模块的信息流与该模块的输出数据之和)。也就是说，模块结构复杂性度量基于程序控制流有向图的拓扑结构。这种度量方法将程序结构表示成有向图，一个模块对应一个节点，节点之间的连接关系就是模块之间的调用关系。模块结构复杂性定义为以起点为顶点的有向图的所有路径数加 1。

模块结构复杂性度量的另一方面是试图反映包括所有模块接口关系在内的整个软件的结构复杂性。模块结构复杂性度量不仅反映模块之间的接口情况，我们还可以通过它将

已发现的失效和软件可靠性及与之相关的数据复杂性有机地联系起来。

对模块结构复杂性的度量，不论是静态的，还是动态的，都是通过对模块或信息的扇入/扇出数量来度量，即通过计算直接进入或流出模块的数据流路径来获得模块的扇入/扇出数据。通常，还可以通过自动数据流层次或 HIPO 图等来确定模块之间、子系统之间或模块与子系统之间信息流的扇入/扇出数量。即接口复杂性可表示成对应模块或子系统的扇入与扇出乘积的平方。

程序只有主模块、无子模块时的结构复杂性为 1。结构复杂性高的软件或模块比结构复杂性低的软件或模块的可靠性差。它将软件复杂性和软件可靠性有机地联系起来。当然，还可以考虑对程序长度加权。

2) 总体复杂性度量

总体复杂性或整个软件结构的复杂性同模块复杂性之间往往相互矛盾。模块划分越小，功能越简单，模块复杂性就越小；但模块之间的联系就越多，接口就越复杂，由此导致的模块结构复杂性就越大。相反，模块复杂性的增加意味着模块结构复杂性的降低。因此，软件总体复杂性是在软件设计尤其软件总体设计中，试图通过对总体复杂性的度量来综合反映软件的功能要求及软件中的应力情况等，并力求平衡模块复杂性与模块结构复杂性；寻找一个合适的复杂性指标，从而达到改善和提高软件总体复杂性的目的，使之最小化。总体复杂性是模块复杂性、模块结构复杂性，以及软件的重要度、调用频率等的综合。但遗憾的是，目前尚没有一种权威的总体复杂性度量方法。

2. 软件复杂性控制

显然，软件复杂性控制可从模块复杂性控制和模块结构复杂性控制两方面进行。

1) 模块复杂性控制

单个模块容易理解，可以分别编制、调试、查错、修改、测试和维护。在容易理解和处理的同时，还可以有效地防止错误蔓延，从而降低软件复杂性，提高软件可靠性。因此，模块化是结构化程序设计的基础，是软件的主要属性，而模块复杂性控制是软件复杂性控制的基础。模块大小由其功能和性能决定，模块化不够或过分模块化都应得到有效控制。模块复杂性控制主要包括模块的大小和结构的控制。

在三种典型的控制结构中，循环结构的复杂性最大。在模块内部，对循环结构复杂性的度量非常有意义。但是，循环复杂性的度量没有将连续程序语句、单独的程序语句或由多支路条件语句引起的控制流复杂性计算在内，这无疑降低了度量的准确性。过度使用相当危险，通过增加模块数量来降低循环复杂性也不理智。

似乎是模块划分越小，由此产生的复杂性就越低，对应的软件可靠性就越高。如果无限地分割软件，最后形成的软件复杂性极小，可靠性水平无限高。而事实上，还有其他因素制约着软件的复杂性。随着模块数目的增加，尽管模块复杂性减小了，但模块之间的接口随之增加，甚至大幅度增加，从而导致结构复杂性增加。因此，应适当地确定模块的大小和接口，使之保持适度的复杂性，这是模块复杂性控制的基本原则。

软件最合适的模块数和最合适的模块大小目前尚无一种精确的指标。因此，在完成软件规定功能的前提下，在开发成本的有效控制下，寻找一种最佳的软件模块大小和数目，

以达到对软件模块复杂性的控制。

为改善模块复杂性，提高软件可靠性，在进行模块设计时需要遵循如下原则：

(1) 抽象化。对于复杂软件的模块化分解，常常采用抽象的层次分解。首先用一些高级的抽象概念进行构造和理解，这些高级的概念又可以用一些较低级的概念来构造和理解，如此进行下去，直至最低层次的具体元素。

(2) 模块化。模块化概念与抽象是一致的。随着软件设计的逐步推进，软件结构中的每个模块层次代表了软件抽象层次的一次精化。事实上，软件结构的顶层模块控制了系统的主要功能，并且影响全局；软件结构的低层模块完成对数据的具体处理。用自顶向下、由抽象到具体的方式分配控制，从而精化软件设计，降低软件的复杂性，提高软件的可理解性、可测试性和可维护性。

(3) 信息隐蔽。在模块功能和大小的确定及设计过程中，遵循信息隐蔽原则不仅有利于改善模块本身的复杂性，也有利于降低软件的结构复杂性。信息隐蔽意味着有效的模块化可以通过定义一组独立的模块来实现，这些独立的模块彼此间仅交换那些为了完成软件功能而必须交换的信息。此外，在软件测试和维护过程中，信息隐蔽也将提供极大的好处。因为大多数数据和过程对软件的其他部分而言是隐蔽的。因此，即使在调试和维护期间由于疏忽而引入了错误，一般也不会波及软件的其他部分。

2) 模块结构复杂性控制

软件结构是软件元素之间的关系表示。软件元素之间的关系多种多样，这些关系均可表示为层次形式，即层次之间是由关系链接的，故受到关系的制约。这种层次结构的概念已经成为软件结构的一种表示形式。

同样为改善模块结构复杂性，提高整个软件可靠性，我们在进行软件体系结构设计时需要遵循如下原则：

(1) 模块独立。开发具有单一功能且和其他模块之间没有更多相互作用的模块，使之保持独立性，是软件结构复杂性控制的基本要求。对模块化程度较高的软件，其功能可以被分割，接口已经简化。因此，这样的软件比较容易编制，尤其适用于不同成员分别编制。而且独立的模块容易测试和维护，因为由设计和代码修改引起的副作用已得到限制，错误不容易蔓延。这样，有可能得到“插件式模块”。总之，模块独立性是优秀软件设计的关键。一般可通过对模块的耦合性和内聚性的控制，来实现对软件结构的控制。

耦合性表示模块之间的相对独立性，是影响软件复杂性的一个重要因素。设计软件时，如果我们遵循尽量使用数据耦合、少用控制耦合、限制公共耦合范围、完全不用内容耦合这 4 条原则，就可能在很大程度上减少模块之间的耦合性，降低模块的复杂性。

内聚是指模块功能的相对强度，用于衡量模块内部各个元素彼此结合的紧密程度，是信息隐蔽和局部化的自然延伸。在软件设计中，虽然没有必要精确地确定软件的内聚性等级，但必须力争使所设计的模块具有高内聚性，并能识别出低内聚性。即适当调整软件设计，以便得到更好的模块独立性。

(2) 适当的扇入/扇出。扇出是影响模块宽度的主要因素，扇出越大，模块越复杂。此时，应适当增加中间层次的控制模块，以降低模块的扇出；而扇出太小时，可把下级模块进一步分解成若干子功能模块或合并到它的上级模块中。因此，模块的扇出不能太大，也

不能太小。经验表明，典型系统的平均扇出通常是 3 或 4，上限是 5 至 9。模块的扇入越大，共享该模块的上级模块数就越多，这是有好处的。但是，不能违背模块独立性原则，而一味地追求高扇入。通常，软件结构对扇入/扇出的要求是，顶层扇出较多，中间层次的扇出应尽量减少，随着深度增加，争取更多的扇入。

(3) 简化软件接口。接口复杂性是产生错误的根源。在设计软件接口时，应尽量使软件接口传递的信息简单，并与模块的功能一致。通常，我们追求的设计是单入口、单出口模块。这样不仅可以有效地防止模块之间的内容耦合，同时也便于降低接口的复杂性和冗余度。此外，还有利于一致性的改善。设计软件接口应尽量避免模块之间的病态连接，杜绝转移进入或引用模块的内部。

3) 软件总体复杂性控制

这是一个系统工程。它是在对软件的各种复杂性度量的基础上，在综合考虑软件开发成本、可靠性要求等一系列因素之后，对软件复杂性的一种平衡。尽管其首要任务是软件总体复杂性控制，但单纯从复杂性来看，可能有时达到的并不是最好的复杂性要求，却可能是一种最优的控制与选择。因此，软件总体复杂性控制不仅是一种技术，更是一门艺术。它需要的不仅是技术、方法和工具，它更需要软件设计人员的超水平智力发挥。

下述内容是形成软件复杂性的重要原因，它们构成了软件复杂性度量的基本度量准则集，是软件复杂性控制的基本出发点：

- 控制结构和数据结构复杂的程序较复杂
- 转向语句使用不当的程序较复杂
- 非局部量较多的程序较复杂
- 按地址调用参数比按值调用参数较复杂
- 模块及过程之间联系密切的程序较复杂
- 嵌套深度越大，程序越复杂
- 循环结构的复杂性大于选择结构和顺序结构的复杂性
- 宽度是软件复杂性的主要形成原因

软件复杂性度量力图对这些准则进行定义和定量描述，找出影响和制约软件复杂性的所有关系。而复杂性控制试图在软件设计中，通过对所有影响软件复杂性，进而影响软件可靠性的因素进行控制，将它们限制在最小的范围内，以改善和提高软件可靠性。另外，在编程语言的选择上，我们可以选择能从软件工程中获取最大好处的现代语言，如 Ada 语言，它能很好地支持抽象化、模块化、信息隐藏以及能够降低耦合、增加内聚的模块独立等。

7.3.2　软件复杂性度量元

软件复杂性度量的度量元很多，主要分为如下几类：

(1) 规模，即总的指令数或源程序行数。

(2) 难度，通常由程序中出现的操作数的数目决定的量来表示。

(3) 结构，通常用与程序结构有关的度量来表示。

(4) 智能度，即算法的难易程度。

软件复杂性直接关系到软件开发费用的多少、开发周期的长短和软件内部潜伏错误的多少。同时它也是软件可理解性的另一种度量。

对软件复杂性进行度量要求满足以下假设：

(1) 它可以用来计算任何一个程序的复杂性。

(2) 对于不合理的程序，例如对于长度动态增长的程序，或者对于原则上无法排错的程序，不应当使用它进行复杂性计算。

(3) 如果程序中的指令条数、附加存储量、计算时间增多，不会减少程序的复杂性。

1. 规模度量元 Line Count 复杂度

基于规模度量程序的复杂性，最简单的方法就是统计程序的源代码行数。此方法基于两个前提：①程序复杂性随着程序规模的增加不均衡地增长；②控制程序规模的最好方法是分而治之，将一个大程序分解成若干个简单的可理解的程序段。

该方法的基本考虑是统计一个程序模块的源代码行数，并以源代码行数作为程序复杂性度量。假设每行代码的出错率为每 100 行源程序中可能有的错误数目，例如每行代码的出错率为 1%，则是指每 100 行源程序中可能有一个错误。

一般程序出错率的估算范围是 0.04%~7%，即每 100 行源程序中可能存在 0.04 至 7 个错误。另外，每行代码的出错率与源程序行数之间不存在简单的线性关系。随着程序的增大，出错率以非线性方式增长。所以，代码行度量法只是一种简单的、估计很粗糙的方法，在实际应用中很少使用。

2. 难度度量元 Halstead 复杂度

Halstead 的软件科学理论也许是最著名和最完全的(软件)复杂度的综合度量，它是软件科学提出的第一个计算机软件分析定律。

Halstead 的软件科学研究确定计算机软件开发中的一些定量规律，它采用以下一组基本度量值，这些度量值通常在程序产生之后得出，或者在设计完成之后估算出。Halstead 根据程序中可执行代码行的操作符和操作数的数量来计算程序的复杂性。操作符和操作数的数量越大，程序结构就越复杂。比如对于代码，我们可以统计它们的操作符和操作数，然后以此为基础，计算程序的长度和体积等。其中，操作符包括程序调用、数学运算符以及有关的分隔符等，操作数可以是常数和变量等。

举例：在高级语言定义中，运算符包括算术运算符、赋值符(=或:=)、逻辑运算符、分界符(，或；或:)、关系运算符、括号运算符以及子程序调用符、数组操作符、循环操作符等；特别地，成对的运算符，例如“begin…end”“for…to”“repeat…until”“while…do”“if…then…else”“(…)”等都当作单一运算符。

假设 n1 表示程序中不同运算符的个数，n2 表示程序中不同操作数的个数，N1 表示程序中实际运算符的总数，N2 表示程序中实际操作数的总数。参见表 7-1。

Halstead 的程序词汇表为一个程序中出现的不同操作符和不同操作数之和：n=n1+n2。

实际的 Halstead 长度，即 N 表示实际的程序长度或简单长度，定义为：N=N1+N2。

我们以 N^表示程序的预测长度，Halstead 给出 N^的计算公式为 $N\hat{}=n1\log_2 n1+n2\log_2 n2$。

Halstead 的重要结论之一是：程序的实际长度 N 与预测长度 N^非常接近，这表明即使

程序还未编写完，也能预先估算出程序的实际长度 N。

表 7-1　FORTRAN 语言程序的操作符、操作数计算举例

	操作符	计数	操作数	计数
SUBROUTINE SORT(X，N)				
DIMENSION X(N)				
IF (N .LT. 2) RETURN	1 语句末	7	1 X	6
DO 20 I=2，N	2 数组下标	6	2 I	5
DO 10 J=1，I	3 =	5	3 J	4
IF (X(I).GE. X(J)) GO TO 10	4 IF( )	2	4 N	2
SAVE=X(J)	5 DO	2	5 2	2
X(I)=X(J)	6 ,	2	6 SAVE	2
X(J)=SAVE	7 程序末	1	7 1	1
10 CONTINUE	8 .LE.	1	n2 = 7	N2 = 22
20 CONTINUE	9 .GE.	1		
RETURN	10 GO TO	1		
END	n1 = 10	N1 = 28		

Halstead 还给出了另外一些计算公式，如下：

(1) 程序容量：$V=N\log_2(n1+n2)$

它表明程序在词汇上的复杂性，其最小值为 $V\hat{}=(2+n2\hat{})*\log_2(2+n2\hat{})V$。这里，2 表明程序中至少有两个运算符：赋值符=和函数调用符 f( )。n2^表示输入/输出变量个数。

对于上面的例子，利用 n1、N1、n2、N2，可以计算得到：

$H=10*\log_2(2\times10)+7*\log_2(2\times7)=52.87$，　$N=28+22=50$，　$V=(28+22)*\log_2(10+7)=204$

等效的汇编语言程序的 V＝328。这说明汇编语言比 FORTRAN 语言需要更多的信息量(以 bit 表示)。

(2) 程序级别：L^=(2/n1) *(n2/N2)

它表明程序的最紧凑形式的程序量与实际程序量之比，反映了程序的效率。其倒数 D=1/L^，表明实现算法的困难程度。

(3) 编制程序所用的工作量：E=V/L^

(4) 智能级别：I=L^*E

(5) 语言级别：L'=L^*L^*V

(6) 编程时间(以小时记)：T^=E/(S*f)。这里 S=60*60，f=18

(7) 平均语句大小：N/语句数

(8) 程序中的错误数目预测值：$B=N\log_2(n1+n2)/3000$

B 为该程序的错误数，它表明程序中可能存在的差错 B 应与程序量 V 成正比。

例如，一个程序对 75 个数据库项共访问 1300 次，对 150 个运算符共使用了 1200 次，那么预测该程序的错误数：

$B=(1200+1300)*\log_2(75+150)/3000\approx6.5$

即预测出该程序中可能包含 6 或 7 个错误。

Halstead 的优点是不用对程序进行深层次的分析，就能够预测错误率，预测维护工作量；有利于项目规划，衡量所有程序的复杂度；计算方法简单；与所用的高级程序设计语言类型无关。众多的研究表明，将 Halstead 复杂度用于预测程序工作计划和程序错误有一定的意义。但 Halstead 复杂度仅考虑程序数据量和程序体积而不考虑程序控制流，不能从根本上反映程序的复杂性。

3. 结构度量元(McCabe 复杂度)

软件复杂性主要表现为软件结构的复杂性，McCabe复杂度是通过对软件结构进行严格的算术分析得来的，实质上是对程序拓扑结构复杂性的度量，明确指出了任务复杂部分。McCabe复杂度包括圈复杂度、基本复杂度、模块设计复杂度、设计复杂度、集成复杂度、行数、规范化复杂度、全局数据复杂度、局部数据复杂度、病态数据复杂度。

在软件工程中，有三种使用 McCabe 复杂性度量的方式：

(1) 作为测试的辅助工具。McCabe 复杂性度量的结果等于通过一个子程序的路径数，因而需要设计同样多的测试案例以覆盖所有路径。如果测试案例数小于复杂性数，则有三种情况：一是需要更多的测试；二是某些判断点可以去掉；三是某些判断点可用插入式代码替换。

(2) 作为程序设计和管理指南。在软件开发中，需要一种简单的方式指出可能出问题的子程序。保持子程序简单的通用方法是设置长度限制，例如 50 行或 2 页，但这实际上是在缺乏测试简明性的有效方法时无可奈何的替代方法。不少人认为 McCabe 度量就是这样一种简明性度量。但是要注意，McCabe 度量数大的程序，不见得结构化就不好。例如，case 语句是简单结构，但可能有很大的 McCabe 度量数(依赖于语句中的分支数)，这可能是由问题和解决方案所固有的复杂性决定的。使用者应当自己决定如何使用 McCabe 度量所提供的信息。

(3) 作为网络复杂性度量的一种方法。Hall 和 Preiser 提出了一种组合网络复杂性度量，用于度量可能由多个程序员在组网过程中按模块化原理建立的大型软件系统的复杂性。

1) McCabe 圈复杂度

McCabe 程序结构的复杂性主要指模块内部程序的复杂性。McCabe 度量方法根据图论和程序结构控制理论，当度量程序结构的复杂性时，首先把程序结构的控制流程图转换为有向图(即程序图)，然后计算强连通有向图的环数来衡量软件的质量。用 McCabe 度量方法得到的复杂度，称为程序结构的圈复杂度。为了方便叙述，我们给出以下几个定义：

定义 1：程序的流程图被简化以后，把程序流程图中每个处理框都退化成一个节点，把流程图中的箭头转换为连接不同点的有向弧，由此得到的有向图称为程序图。

定义 2：强连通图是指从图中任何一个节点出发都能到达所有的其他节点。

定义 3：强连通有向图的环数是指在一张强连通有向图中线性无关环的个数，即有向图 G 中的弧数 m 与节点数 n 的差再加上分离部分的数目 p，计算公式为 $V(G)=m-n+p$。

圈复杂度除了上述计算方法外，还有其他方法，例如：圈复杂度等于程序图中判定节点的数目加 1；圈复杂度等于强连通程序图在平面上围成的区域个数。

图 7-6 是一幅把考试成绩达到 80 的学生的学号和成绩打印出来的程序控制流图，把它转换为程序图后，如图 7-7 所示。

从图 7-7 中可知其不是强连通图，需要从节点 6 到节点 1 添加一条虚弧线。由此，程序图中有 9 条弧，节点数为 7，根据公式 V(G)=m−n+p 可得出圈复杂度为 3；程序图中判定节点是 3 和 5，用判定节点数目加 1，同样可以得到圆复杂度为 3；加上新添的虚弧线，根据程序图中围成的区域个数，同样也可以得到圈复杂度为 3。

圈复杂度在数量上表现为独立路径的条数，即合理地预防错误所需测试的最少路径条数，圈复杂度大说明程序代码可能质量低且难于测试和维护，经验表明，程序的可能错误和高的圈复杂度有着很大关系。

McCabe圈复杂度的优点：①避免软件中的错误倾向；②指出极复杂模块，这样的模块也许可以进一步细化；③度量测试计划，确定测试重点；④在开发过程中通过限制程序逻辑，指导测试过程；⑤指出将要测试的区域；⑥帮助测试人员确定测试和维护对象；⑦与所用的高级程序设计语言类型无关。

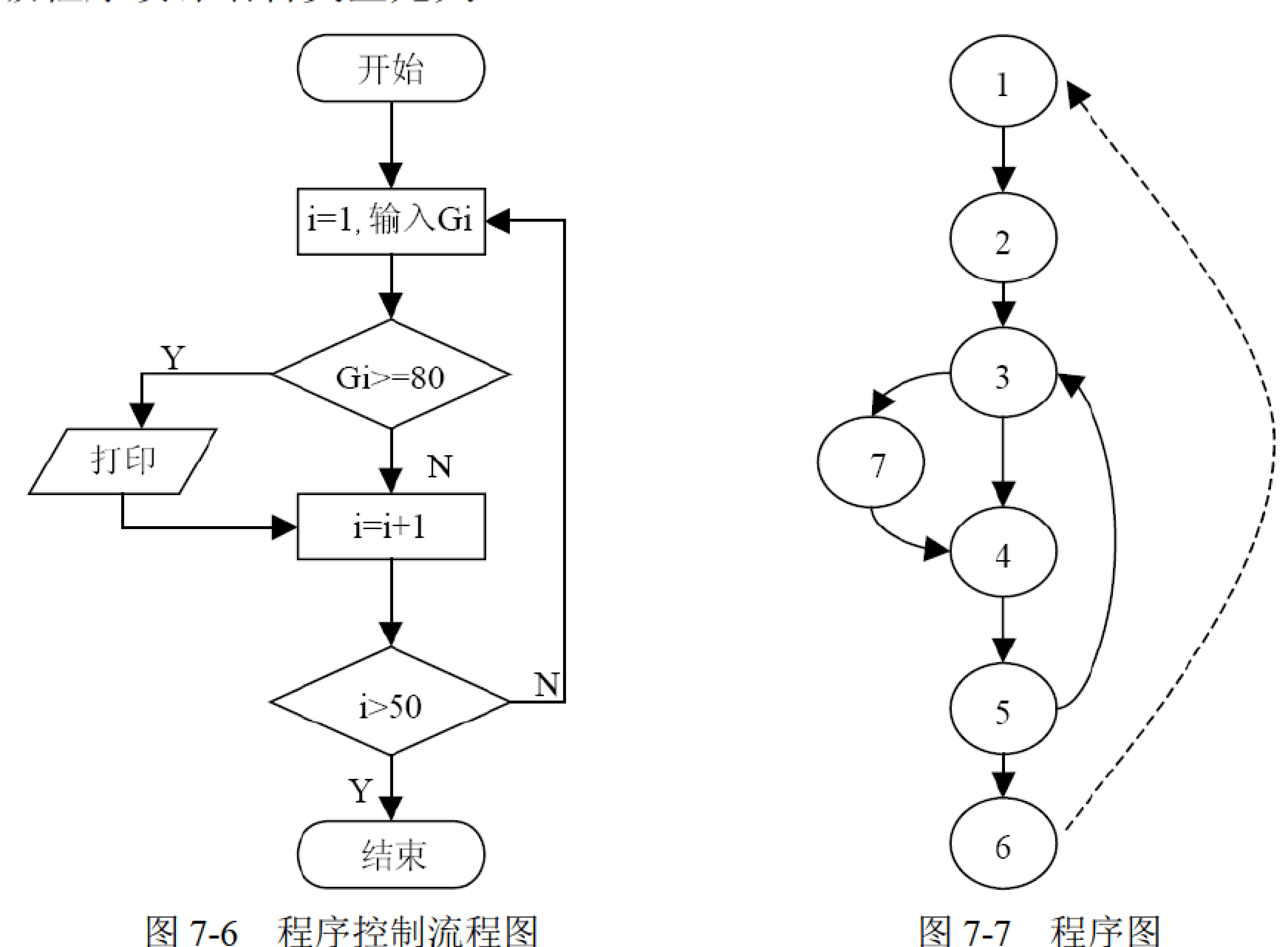


图 7-6　程序控制流程图　　图 7-7　程序图

McCabe圈复杂度的应用：圈复杂度指出了为确保软件质量应该检测的最少基本路径的数目。在实际中，测试每一条路径是不现实的，测试难度随着路径的增加而增加。但测试基本路径对衡量代码复杂度的合理性是很有必要的。

实践表明，模块的圈复杂度，即模块复杂性和模块中存在的软件错误数或缺陷数，以及为了发现并改正它们所需的时间之和，存在一种明显的关系。McCabe指出，V(G)可用作最大模块复杂性的定量指标。通过大量软件工程数据研究发现，圈复杂度到 10 是模块复杂性的实际上限，当V(G)超过这个值时，因为高的圈复杂度使测试变得更加复杂而且增大了软件错误产生的概率。

McCabe圈复杂度提示：圈复杂度度量是测量软件模块中的分支数目，在所有的开发周期中都要使用。

圈复杂度度量以软件的结构流程图为基础。控制流程图描述了软件模块的逻辑结构。模块在典型的语言中是一个函数或子程序，有一个入口和一个出口，也可以通过调用/返回机制设计模块。软件模块的每条执行路径，都有与从模块的控制流程图的入口到出口的节

点相符合的路径。

McCabe的圈来源于非直接连接基本测试周期的数目，更重要的是，也通过直接相连的图表给出独立路径的数目。通过图表的相关性，一个节点可到达另一个节点。

圈复杂度度量也可作为模块基本流程图路径的数目，其重点在于将模块线性组合后，所产生的路径数目是最少的。

对圈复杂度的限制：现在有许多好方法可以用来限制圈复杂度。过于复杂的模块容易出错，难于理解、测试、更正，所以应当在软件开发的各个阶段有意识地限制复杂度，许多开发者已经成功地实现把对软件复杂度的限制作为软件项目的一部分，尽管在确切的数目上略微有些争议。最初支持的数目是 10，现在支持的数目可达 15。但是，只应当在条件较好的情况下使数目大于 10，例如开发者非常有经验，设计合乎正式标准，使用现代化的程序语言、结构程序、代码预排和先进的测试计划。换句话说，开发团队可以选择超过 10 的限制数目，但是必须根据经验进行一些取舍，把精力花在比较复杂的模块上。

McCabe 度量方法反映了代码的质量；预测代码维护量；辅助模块划分；与所有的高级程序设计语言类型无关。但是其实质是对程序控制流复杂性的度量，并不考虑数据流，因而其科学性和严密性具有一定的局限性。

1) Essential Complexity (ev(G))基本复杂度

基本复杂度用来衡量程序的非结构化程度，非结构成分降低了程序的质量，增加了代码的维护难度，使程序难于理解。因此，基本复杂度高意味着非结构化程度高，难以模块化和维护。实际上，消除了一个错误有时会引起其他的错误。

基本复杂度的计算方法是将圈复杂度图中的结构化部分简化成一个点，计算简化以后流程图的圈复杂度就是基本复杂度。其优点是：衡量非结构化程度；反映代码的质量；预测代码维护量，辅助模块划分；与所用的高级程序设计语言类型无关。它的应用方法是：当基本复杂度为 1 时，这个模块是充分结构化的；当基本复杂度大于 1 而小于圈复杂度时，这个模块是部分结构化的；当基本复杂度等于圈复杂度时，这个模块是完全非结构化的。

2) Module Design Complexity (iv(G))模块设计复杂度

模块设计复杂度用来衡量模块判定结构，即模块和其他模块的调用关系。软件模块设计复杂度高意味模块耦合度高，这将导致模块难以隔离、维护和复用。

模块设计复杂度的计算方法是从模块流程图中移去那些不包含调用子模块的判定和循环结构后得出的圈复杂度，因此模块设计复杂度不能大于圈复杂度，通常远小于圈复杂度。模块设计复杂度用于：衡量模块对其下层模块的支配作用；衡量一个模块到其子模块进行集成测试的最小数量；定位可能多余的代码；以复杂的计算逻辑和设计来区分模块；是设计复杂度(S0)和集成复杂度(S1)计算的基础；与所用的高级程序设计语言类型无关。

3) Design Complexity (S0)设计复杂度

设计复杂度以数量来衡量程序模块之间的相互作用关系，提供了系统级模块设计复杂度的概况，有助于衡量进行自底向上集成测试的效果，而且提供了全面衡量程序设计规格和复杂度的数据，不反映独立模块的内部情况。高设计复杂度的系统意味着系统各部分之间有着复杂的相互关系，这样系统将难以维护。

S0 是程序中所有模块设计复杂度之和，可应用于完整的软件，也可应用于任何子系统；

衡量代码的质量；指出模块整体的复杂度，反映每个模块与其内部模块的控制关系；揭示程序中模块调用的复杂度；有助于集成复杂度的计算。

4) Integration Complexity (S1)集成复杂度

集成复杂度是为了防止错误所必须进行的集成测试的数量表示，另一种说法是程序中独立线性子树的数目，一棵子树是一个有返回值的调用序列。就像圈复杂度是测试路径的数目，而集成复杂度是程序或其子系统的独立线性子树。

程序的集成复杂度和模块的圈复杂度是非常相似的，必须计算对程序进行完全集成测试所需测试独立线性子树的数目。集成复杂度 S1 的计算方法是：S1=S0−N+1，N 是程序中模块的数目。

集成复杂度计算有助于集成测试的实施；量化集成测试工作且反映了系统设计复杂度；有助于从整体上隔离系统复杂度。

5) McCabe 的其他复杂度计算

McCabe 的 Number of Lines (nl)行数是模块中总的行数，包括代码和注释。

McCabe 的 Normalized Complexity (nv)规范化复杂度是圈复杂度除以行数，与所用的高级程序设计语言类型无关；定义那些有着显著判定逻辑密度的模块，这些模块相对于其他常见规范模块需要做更多的维护工作。

McCabe 的 Global Data Complexity (gdv(G))全局数据复杂度量化了模块结构和全局数据变量的关系，说明了模块对外部数据的依赖程度，同时度量了全局数据的测试工作，也描述了模块之间的耦合关系，能反映潜在的维护问题。

McCabe 的 Specified Data Complexity (sdv(G))局部数据复杂度量化了模块结构和用户局部数据变量的关系，同时度量了局部数据的测试工作。

McCabe 的 Pathological Complexity (pv(G))病态数据复杂度衡量模块包含的完全非结构化成分的程度，标出向循环内部跳入的问题代码，而这些部分具有最大的风险度，通常需要重新设计。其计算方法是：所有的非结构部分除去向循环内跳入的结构，转换为线结构，病态复杂度就等于简化以后流程图的圈复杂度。它与所用的高级程序设计语言类型无关，指出了可靠性的问题，降低了维护风险，帮助人们识别极不可靠的软件。

4. 其他度量元

除了前面介绍的 Line Count、Halstead 和 McCabe 三种复杂性度量外，对模块复杂性、模块结构复杂性进行度量还有很多度量元，如函数参数个数、路径数、层次数、直接调用个数、return 语句个数、调用者的个数、goto 语句个数、词汇频度、局部变量个数、注释率、函数中的可执行语句数、宏定义个数、扇入/扇出数等。

7.3.3　面向对象的软件复杂性度量

随着 OO(Object-Oriented，面向对象)技术和工具的发展日益成熟，OO 设计显示出了其巨大的优越性。各种运用 OO 技术分析、设计的软件系统越来越多，传统的软件度量方法无法适应 OO 技术中采用的数据抽象、封装、继承、多态性、信息隐藏、重用等机制，这些机制在传统的面向过程软件开发中是缺乏的，因此，运用传统的度量方法不能很好地

反映 OO 技术的特征，例如：继承提供的重用，属于对象自身的代码较少，如果用源代码行(LOC)来度量整个对象，显然结果是不能令人满意的，如果仅把类看作模块，也是不恰当的，因为功能性模块之间的耦合关系表现在接口上，主要是参数传递、对全程变量的访问以及模块间的调用关系，而对象间的耦合关系主要表现为通过继承、通过消息传递和接收、通过对抽象数据类型的引用带来的耦合。传统的度量方法无法反映这些特征，因此，需要额外的度量方法(即面向对象度量方法)来反映这些关系。

与传统软件一样，OO 度量的主要目的是：更好地理解产品的质量，评价过程的效率，改进项目完成工作的质量。

1. 面向对象度量的特性

任何产品的技术度量都取决于产品的特性。OO 软件与使用传统方法开发的软件的度量方法截然不同，OO 软件目前常用的五个特殊度量的特性包括以下几个：

1) 局域性

局域性(Localization)是指信息被集中在一个程序内的方式。

(1) 传统方法：数据与过程分离，功能分解和功能信息局域化。其典型的实现形式为过程模块，工作时用数据驱动功能。

此时的度量放在功能内部结构和复杂性上(如模块规模、聚合度、环路复杂性等)或放在该功能与其他功能(模块)的耦合方式上。

(2) OO 方法：局域性基于对象，因为类是 OO 系统的基本单元，对象封装数据和过程，因此应把类(对象)作为一个完整实体来量度。

另外，操作(功能)和类之间的关联是一对一的。因此，在考虑类合作中的度量时，必须能适应一对多和多对一关联。

2) 封装性

封装性(Encapsulation)是指一个项集合的包装。

(1) 传统方法：记录、数组，只有数据没有过程，为低层次的封装；过程、函数、子例程和段，则只有过程没有数据，为中层次的封装。其度量的重点分别在代码行的数据和环路的复杂性。

(2) OO 方法：OO 系统封装拥有类的职责(操作)，包括类的属性、操作和特定的类属性值定义的类(对象)的状态。其度量和重点不是单一的模块，而是包含数据(属性)和过程(操作)的包。

3) 信息隐藏

信息隐藏(Information hiding)是指隐藏(删除)了程序构件操作的细节，只将访问该构件所必要的信息提供给访问该构件的其他构件。

在这一点上，OO 方法和传统方法基本一致。因此，OO 系统应支持信息隐藏，除提供隐藏等级说明的度量外，还应提供 OO 设计质量指标。

4) 继承性

继承性(Inheritance)是指一个对象的属性和操作能够传递给其他对象的机制。继承性的发生贯穿于一个类的所有层次。

一般来说，传统软件不支持这种特性。而对于 OO 系统来说，继承性是一个关键的特性。因此，很多 OO 系统的度量都以此为重点，如子的数量(类的直接实例数量)、父的数量(直接上一代数量)，以及类的嵌套层次(在一个继承层次中，类的深度)。

5) 抽象

抽象(Abstraction)使设计者只关心程序构件的主要细节(数据和过程两者)，而不考虑底层的细节。

抽象也是一种相对概念，抽象在 OO 和传统开发方法中都被采用，如处于抽象的较高层次时，我们可忽略更多的细节，只提供关于概念或项的一般看法；当处于抽象的较低层次时，可以引入更多的细节，即提供关于概念或项的更详细看法。

在 OO 中，由于类是一个抽象，它可以从许多不同的细节层次和用许多不同的方式(如作为一个操作的列表、一个状态的序列、一组合作)来观察。

OO 度量可用一个类度量的项来表示抽象，如每个应用类被实例化的数量、每个应用类被参数化的数量，以及类被参数化与未被参数化的比例等。

2. 面向类的度量

类是 OO 系统的基本单元。对单一类、类的层次和合作的度量，对于必须评价设计质量的面向对象软件来说是十分重要的。

第一批以类为对象开展面向对象度量方法研究的是 Chidamber 和 Kemerer(C&K 或 CK)。CK 度量是使用最为广泛的度量体系之一，他们建议使用 6 种基于类设计的度量(通称为 CK 度量组)：①每个类的加权方法数；②继承树的深度；③子类的个数；④对象类之间的耦合；⑤类的响应；⑥方法中的聚合缺乏。

与此同时，Lorenz 和 Kidd 提出了 LK 度量组，他们把基于类的度量分为四种类型：规模、继承、内部(特性)和外部(特性)。对 OO 类进行基于规模的度量，主要集中在单一类的属性和操作的数量上，以及作为整个 OO 系统的平均值；基于继承的度量，关注的是贯穿类层次的操作被重用的方式；类的内部特性的度量是考查聚合和代码问题；而外部特性的度量则是检查耦合和重用问题。

由于 CK 度量方法和 LK 度量方法的出发点是类这一级，虽对系统开发有帮助，却没有提供对系统的判断。为此，Abrito 等人针对面向对象属性提出了一套称为 MOOD 的度量。MOOD 度量则是系统级的，它从封装性、继承性、耦合性和多态性四方面提出了度量指标。

3. 面向操作的度量

因为类是 OO 系统设计中的基础框架或模板设施，而 OO 系统真正活动的实体是类的对象。类的对象具有状态和行为两大特性，分别用数据和操作表示。因此对类操作进行复杂性度量是很有意义的。根据 Lorenz 和 Kidd 的建议，对类操作进行复杂性度量有三种度量元。

(1) 平均操作规模 OSavg(Average Operation Size)

虽然代码行数可以成为操作规模的指示器，但代码行数的度量与传统软件一样有许多问题。因此，在 OO 软件中，由操作传送的消息数量提供了对操作规模度量可选的方法。操作规模增大，表示操作所传送的消息数量增加，数量越大，说明越复杂。

(2) 操作复杂性 OC(Operation Complexity)

操作的复杂性可以用传统软件所使用的任何复杂性量度进行计算。因为操作限于特定的职责，所以设计人员应使 OC 尽可能小。

(3) 每个操作参数的平均数 NPavg(Average Number per Operation)

操作参数的数量越大，对象间的合作就越复杂。所以，NPavg 一般应尽可能小。

这里所说的三种面向操作的度量列出了对 OO 操作可提供的几个度量元，通过对操作的度量来评价操作、操作传递的消息及对象间协作的复杂性，获得这些度量结果，可以对整个系统的操作的复杂程度有所了解，但并不能代表整个软件系统的复杂度，还需要从多个角度综合考虑。

4. 面向 OO 系统的度量

根据 Binder 的建议，按照对封装性和继承性的影响提出了几类面向 OO 系统的度量方法。

1) 封装性

(1) 方法(操作与服务)中聚合的缺乏(LCOM)

LCOM 的值越高，表示更多的状态必须进行测试，才能保证方法不产生副作用。

(2) 公共与私有的百分比(PAP，Percent Public and Protected)

公共属性从其他类继承，所以这些类是可见的。私有属性是专属的，为一特定子类拥有。这种量度说明类的公共属性的百分比，PAP 的值高就可能增加类间的副作用。

(3) 数据成员的公共访问(PAD，Public Access to Data Member)

这种量度说明可访问其他类属性的类的数量，即一种封装的破坏。PAD 的值高可能导致类间的副作用。所以，测试的设计必须保证每一种这样的副作用能够被发现。

2) 继承性

(1) 根类的数量(NOR，Number of Root Classes)

这种量度是在设计模型中，描述性质各不相同的类层次数量的计算方法。对于每一个根类及其子类的层次必须开发相应的测试序列。随着NOR的增大，测试工作量也相应增加。

(2) 扇入(FIN，Fan In)

当扇入用于 OO 情况时，扇入是一种多重继承的指标。FIN 大于 1，说明一个类不只从属于一个根类，而是继承更多的根类的属性和操作。

(3) 子类的数量(NOC，Number of Children)和继承树的深度(DIT，Depth of the Inheritance Tree)

父类的方法(操作、服务)发生变化将导致需要对每个子类进行重新测试。

7.4　软件质量模型

软件质量是软件工程开发工作中的关键问题，也是软件工程生产中的核心问题。因为软件质量不高是导致软件项目进度延误、预算超支或项目失败、项目终止等软件危机的根

本原因。另外，软件质量高可以降低项目开发的总成本，例如：降低维护成本(并延长软件的生命周期)，降低软件失效导致的成本损失。最后，我们可以通过减少并纠正实际的软件开发过程和软件开发结果与预期的软件开发过程和软件开发结果的不符合情况，通过在软件开发周期中尽可能早地预期或检测到不符合的情况，防止错误发生，减少错误纠正成本等手段来保证软件质量。但在开展这些工作之前，我们必须回答什么是软件质量。

7.4.1　软件质量的概念

软件质量是一种模糊而又捉摸不定的概念。我们在日常生活中常听到的关于软件评价有：这个软件真好用；这个软件功能全面、结构合理、简单易用。这些很平常的语言根本不能算作软件质量的评价，尤其不能算作软件质量科学的定量评价。

不同的人从不同的角度来看软件质量问题，会有不同的理解。从用户的角度看，质量就是满足客户的需求；从开发者的角度看，质量就是与需求说明保持一致；从产品的角度看，质量就是产品的内在特点；从价值的角度看，质量就是客户是否愿意购买。

现代质量管理认为，质量是客户要求或期望的有关产品或服务的一组特性，落实到软件上，这些特性可以是软件的功能、性能和安全性等。这些特性决定了软件产品保证客户满意的能力，并且这些特性应该是可以度量的。虽然软件的无形性和复杂性使得软件质量的度量要比其他产品(比如电视机)困难得多，但我们仍可以借助软件测试的理论、技术、方法和工具来获得软件质量客观、科学的度量。

另外，我们还可以从另一个角度，即软件产品是如何生产出来的，来间接地推断软件质量。我们称之为软件流程质量，以有别于前面所说的软件产品质量。所谓流程，我们可以将其理解为一个活动序列和与此相关的输入、输出、约束条件、实现方法、辅助工具等因素共同组成的系统。ISO 9001 和 SW-CMM 都主要是从流程角度来探讨软件质量和质量改进的。

当然，我们还能从其他角度(比如软件的生产者-人的素质)来诠释软件质量，但不管怎样，软件产品质量是最终的检验标准，而最终的检验者就是客户。从这个意义上说，软件质量就是客户满意度。但这种说法太笼统，不好掌握。为此，我们必须给软件质量一个明确的概念或定义，以帮助我们以一个明确的尺度来检验质量。

1. 软件质量的定义

国际标准化组织 ISO 在质量特性国际标准 ISO/IEC 9126 中将软件质量定义为反映软件产品满足规定需求和潜在需求能力的特征和特性的总和。MJ.Fisher 将软件质量定义为：所有描述计算机优秀程度的特性的组合。也就是说，为了满足软件的各项精确定义的功能、性能要求，符合文档化的开发标准，需要相应地给出或设计一些质量特性及其组合，要得到高质量的软件产品，就必须使这些质量特性得到满足。

按照 ANSI/IEEE Std 1061-1992 中的标准，软件质量的定义为：与软件产品满足需求所规定的和隐含的能力有关的特征或特性的全体。具体包括：

(1) 软件产品中所能满足用户给定需求的全部特性的集合。

(2) 软件具有所有的各种属性组合的程度。

(3) 用户主观得出的软件是否满足其综合期望的程度。

(4) 决定所用软件在使用中将满足其综合期望程度的合成特性。

通过类比，我们这样理解软件质量：软件质量是许多质量属性的综合体现，各种质量属性反映了软件质量的方方面面。人们通过改善软件的各种质量属性，从而提高软件的整体质量(否则无从下手)。

2. 软件质量特性

对于软件质量有三种不同的视角。用户主要感兴趣的是如何使用软件、软件性能和使用软件的效果。所以他们关心的是：①是否具有所需要的功能；②可靠程度如何；③效率如何；④使用是否方便；⑤环境开放的程度如何(即对环境、平台的限制，与其他软件连接的限制)。

而开发者负责生产出满足质量要求的软件，所以他们关心的是中间产品的质量以及最终产品。对于管理者来说，更注重总的质量，而不是某一特性。

从管理角度对软件质量进行度量的那些影响软件质量的主要因素，可划分为三组，分别反映用户在使用软件产品时的三种观点。即：①正确性、健壮性、效率、完整性、可用性、风险(产品运行)；②可理解性、可维修性、灵活性、可测试性(产品修改)；③可移植性、可重用性、互运行性(产品转移)。

而按照 ISO/IEC 9126 的规定，软件质量可用 6 个特性来评价：

(1) 功能性(functionality)是与一组功能及指定的性质有关的一组属性。这里的功能是指满足明确或隐含的要求的那些功能，而这组属性以软件为满足需求做些什么来描述，而其他属性则以何时做和如何做来描述。

(2) 可靠性(reliability)是与在规定的一段时间和条件下，软件维持其性能水平的能力有关的一组属性。在这里要强调的是：软件不会老化。因此可靠性的种种局限是由于需求、设计和实现中的错误所致，由这些错误引起的故障取决于软件产品使用方式和程序任选项的选用方法，而不取决于时间的流逝。

(3) 可用性(usability)是与一组规定或潜在用户为使用软件所需付出的努力和对这样的使用所做评价有关的一组属性。这里软件的可用性主要是指人们学习、操作、准备输入和解释程序输出(输出结果和出错信息)的便利程度，反映了与用户的友善性，即用户在使用本软件时是否方便。

(4) 效率(efficiency)是指在规定条件下，相对于所用资源数量，软件产品提供适当性能的能力。效率反映了在完成功能要求时，有没有浪费资源，资源这个术语有比较广泛的含义，包括内存、外存的使用，还有通道能力及处理时间。

(5) 维护性(maintainability)是指软件产品可被修改的能力。修改包括为了适应环境的变化和需求、功能规格说明的变化而对软件进行的修改、改进或更改。可维修性反映了在用户需求改变或软件环境发生变更时，对软件系统进行相应修改的容易程度。

(6) 可移植性(portability)是指软件产品从一个计算机系统/环境转移到另一个计算机系统/环境的容易程度。这里的环境包括系统体系结构环境、硬件或软件环境。

显然，软件质量所反映的问题包括如下几个方面：

(1) 软件需求是度量软件质量的基础。

(2) 软件人员必须用工程化的方法来开发软件，否则软件质量就得不到保证。要根据

指定的标准来定义一组指导软件开发的准则。如果不能遵守这些准则，就极有可能导致质量不高。

(3) 软件要满足一些隐含的需求，如可维护性、可靠性等。

总之，计算机软件质量是计算机软件内在属性的组合，包括计算机程序、数据、文件等多方面的可理解性、正确性、可用性、可移植性、可维护性、可修改性、可测试性、灵活性、再用性、完整性、适用性、健壮性、可靠性、效率与风险等多方面特性，是与软件产品满足规定的和隐含的需求的能力有关的特性和特性的全体，包括明确声明的功能和性能需求，明确文档化过程的开发标准，而且由专业人员开发的软件应具有的所有隐含特性都得到满足。对于高质量的软件，要能够按照预期的时间和成本提交给用户，并能够按照预期要求正确工作。

7.4.2　软件质量分层模型

尽管软件质量是难以定量度量的软件属性，但人们还是想尽办法来定性和定量地描述软件质量。依上所述，软件质量可用特定的软件质量特性来表示，而软件质量特性反映了软件的本质。讨论软件的质量，问题最终要归结到定义软件的质量特性；而定义软件的质量，就等价于为软件定义一系列质量特性。软件质量特性是面向管理的观点，或是从使用者的观点引入，体现了用户想要什么样的软件。

人们通常用软件质量模型来描述影响软件质量的特性。已有多种有关软件质量的模型。它们共同的特点是基于软件质量特性定义软件质量分层模型。一般是定义为三层模型，参见图 7-8。在这三层模型中，软件质量不仅从软件外部表现出来的特性来确定，而且必须从其内部所具有的特性来确定；第二层的质量子特性是上层质量特性的细化，一个特定的子特性可以对应若干个质量特性，软件质量子特性是管理人员和技术人员关于软件质量问题的通信渠道；下层是软件质量度量因子或度量元(包括各种参数)，用来度量质量特性。定量化的度量元可以直接测量或统计得到，为最终得到软件质量子特性值和特性值提供依据。

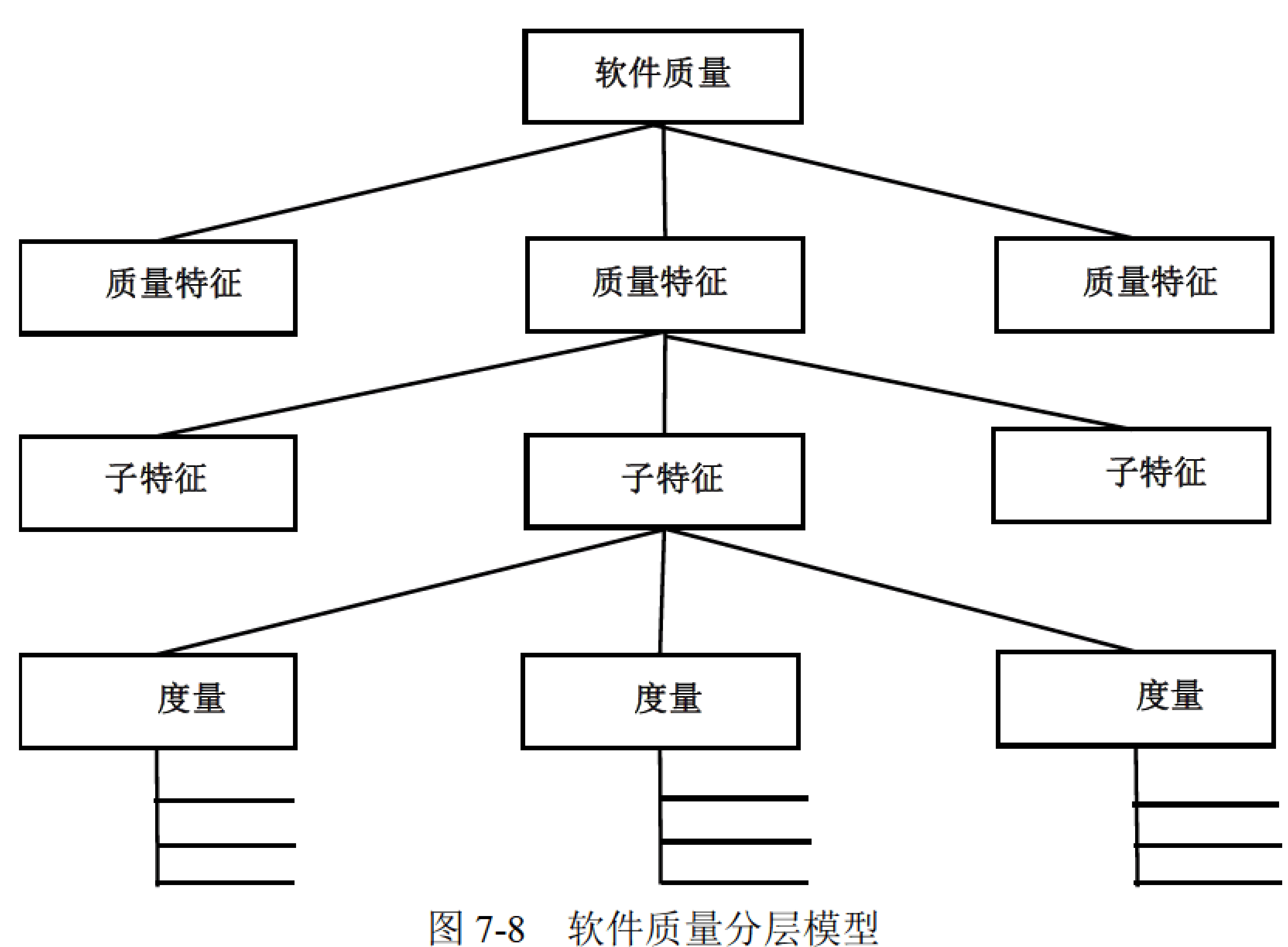


图 7-8　软件质量分层模型

目前，已有很多质量模型，它们分别定义了不同的软件质量属性。比较常见的三个质量模型是 McCall 模型(1977 年)、Boehm 模型(1978 年)、ISO/IEC 9126 模型(1993 年)和 GB/T 16260-2006 模型(2006 年)。下面介绍几个影响较大的软件质量模型。

1. McCall 质量模型

McCall 等人将软件质量分解至能够度量的层次，提出 FCM 三层模型，即软件质量要素(factor)、衡量标准(criteria)和量度标准(metrics)，见表 7-2。

表 7-2　McCall 的 FCM 三层质量模型

层　级	名　称	内　容
第一层	质量要素：描述和评价软件质量的一组属性	功能性、可靠性、可用性、效率性、可维护性、可移植性等质量特性以及将质量特性细化产生的副特性
第二层	衡量标准：衡量标准的组合反映某一软件质量要素	精确性、稳健性、安全性、通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、文件完备性等
第三层	量度标准：可由各使用单位自定义	根据软件的需求分析、概要设计、详细设计、编码、测试、确认、维护与使用等阶段，针对每一个阶段制定问卷表，以此实现软件开发过程的质量度量

在 FCM 三层模型中，软件质量概念基于 11 个基本特性之上，而这 11 个基本特性分别面向软件产品的产品操作(product operation)、产品修正(product revision)和产品转移(product transition)。这 11 个基本特性分别是：

(1) 正确性：一个程序满足它的需求规约和实现用户任务目标的程度。

(2) 可靠性：一个程序满足所需的精确度以完成它的预期功能的程度。

(3) 有效性：一个程序完成其功能所需的计算资源和代码的度量。

(4) 完整性：对未授权人员访问软件或数据的可控制程度。

(5) 可用性：学习、操作、准备输入和解释程序输出所需的工作量。

(6) 可维护性：定位和修复程序中的一个错误所需的工作量。

(7) 灵活性：修改一个运行的程序所需的工作量。

(8) 可测试性：测试一个程序以确保它完成所期望的功能所需的工作量。

(9) 可移植性：把一个程序从一个硬件和/或一个软件系统环境移植到另一个环境所需的工作量。

(10) 可复用性：一个程序可以在另一个程序中复用的程度。

(11) 互操作性：连接一个系统和另一个系统所需的工作量。

McCall 将这 11 类软件质量特性分为三类质量要素，它们之间的关系如图 7-9 所示。

第一类质量要素表现软件的运行特征，包括正确性、可靠性、有效性、完整性和可用性。第二类质量要素表现软件承受修改的能力，包括可维护性、灵活性、可测试性。第三类质量要素表现软件对新环境的适应程度，包括可移植性、可重用性、可互操作性。

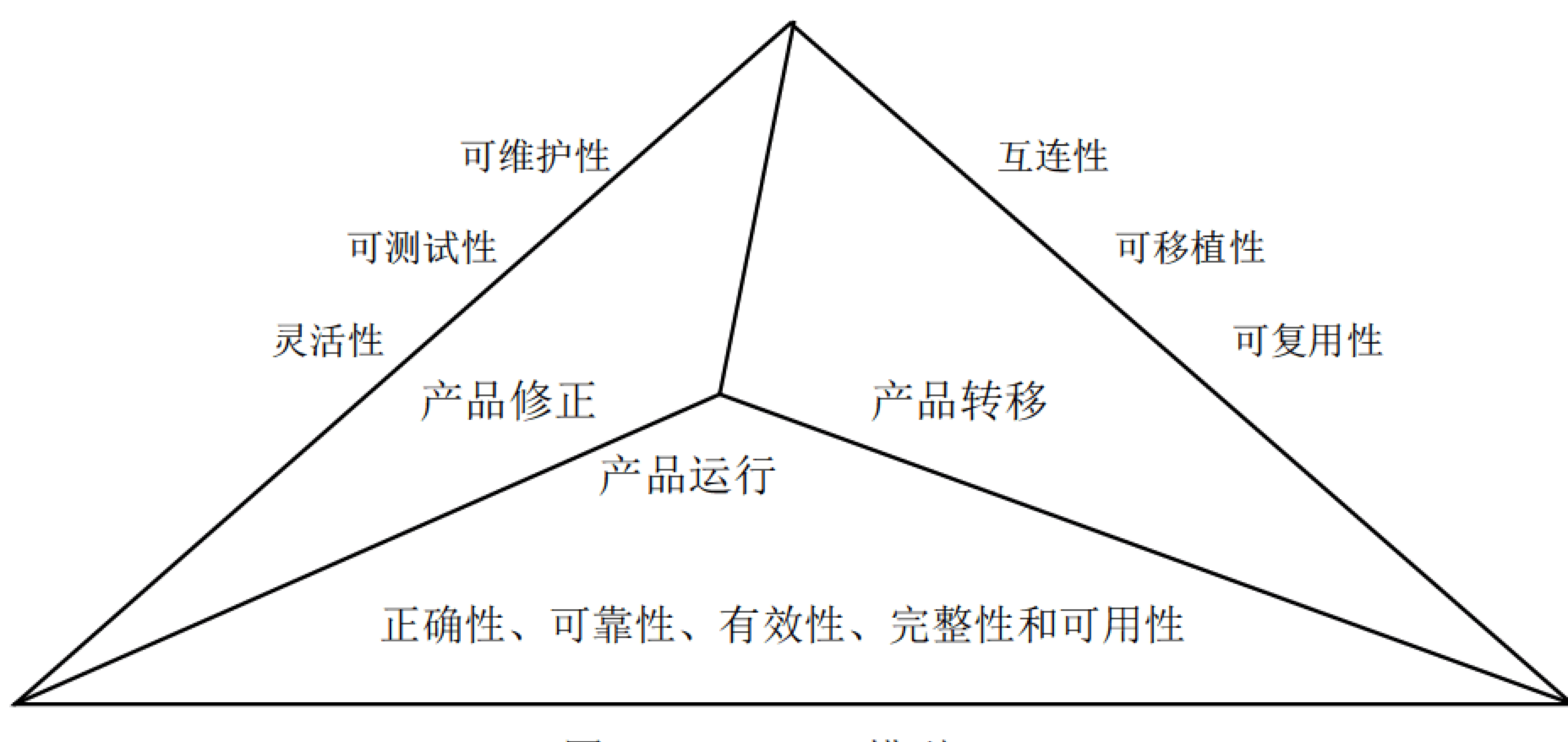


图 7-9　McCall 模型

McCall 等人在 FCM 三层模型的基础上又给出了一个三层次模型的质量度量框架，如图 7-10 所示。McCall 等人认为，要素是软件质量的反映，软件属性可用作评价的准则，定量地度量软件属性，进而可知软件质量的优劣。

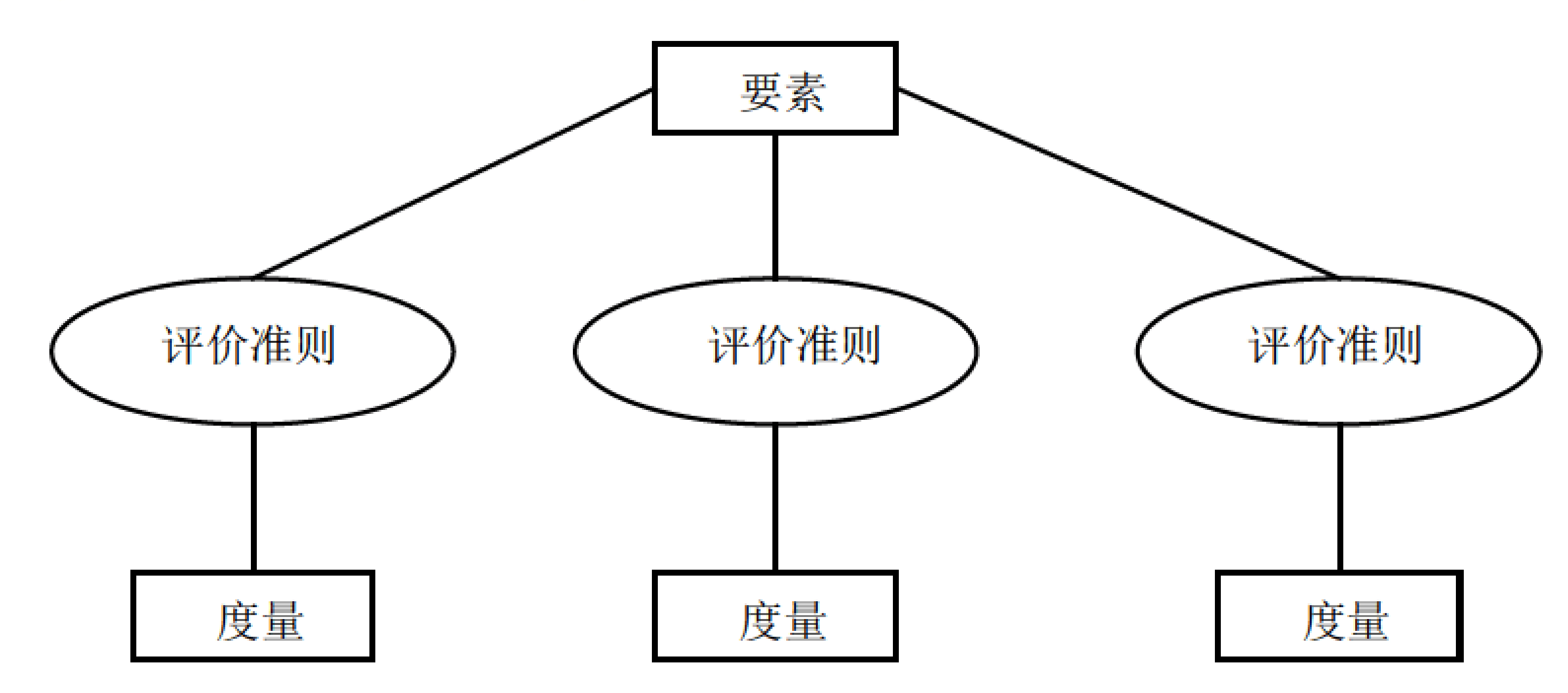


图 7-10　McCall 度量模型框架

McCall 定义的软件质量要素评价准则共 21 种，它们是：

(1) 可审查性(audit ability)：检查软件需求、规格说明、标准、过程、指令、代码及合同是否一致的难易程度。

(2) 准确性(accuracy)：计算和控制的精度，是对无误差程序的一种定量估计，最好表示成相对误差的函数。值越大表示精度越高。

(3) 通信通用性(communication commonality)：使用标准接口、协议和频带的程度。

(4) 完全性(completeness)：软件系统不丢失任何重要成分，完全实现系统所需功能的程度。

(5) 简明性(conciseness)：程序源代码的紧凑性。

(6) 一致性(consistency)：在软件开发项目中一致的设计和文档技术的使用。

(7) 数据通用性(data commonality)：在程序中使用标准的数据结构和类型。

(8) 容错性(error-tolerance)：系统在各种异常条件下提供继续操作的能力。

(9) 执行效率(execution efficiency)：程序运行效率。

(10) 可扩充性(expandability)：能够对结构设计、数据设计和过程设计进行扩充的程度。

(11) 通用性(generality)：程序部件潜在的应用范围的广泛性。

(12) 硬件独立性(hardware independence)：软件同支持其运行的硬件系统不相关的程度。

(13) 检测性(instrumentation)：监视程序的运行，一旦发生错误，就标识错误的程度。

(14) 模块化(modularity)：程序部件的功能独立性。

(15) 可操作性(operability)：操作一个软件的难易程度。

(16) 安全性(security)：控制或保护程序和数据不受破坏的机制，以防程序和数据受到意外的或蓄意的存取、使用、修改、毁坏或泄密。

(17) 自文档化(self-documentation)：源代码提供有意义文档的程度。

(18) 简单性(simplicity)：理解程序的难易程度。

(19) 软件系统独立性(software system independence)：程序与非标准的程序设计语言特征、操作系统特征以及其他环境约束无关的程度。

(20) 可追踪性(traceability)：从设计表示或实际程序构件中学到需求的能力。

(21) 易培训性(training)：软件支持新用户使用系统的能力。

2. Boehm 质量模型

Boehm 在 1976 年首次提出了软件质量层次模型，认为软件产品的质量基本上可从软件的可用性、可维护性和可移植性三个方面来考虑，并将软件质量在概念上分解为若干层次，对于最低层软件质量概念引入量化指标，以便得到软件质量的整体评价。

Boehm 在软件质量层次模型中的第一层，同样给出了功能性、可靠性、可用性、效率、可维护性和可移植性 6 个质量特性。

Boehm 在软件质量层次模型中的第二层给出了 22 个软件质量评价准则：精确性(在计算和输出时所需精度的软件属性)、健壮性(在发生意外时，能继续执行和恢复系统的软件属性)、安全性(防止软件受到意外或蓄意的存取、使用、修改、毁坏或泄密的软件属性)， 以及通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、产品文件完备性。

Boehm 质量模型第一层与第二层的关系如图 7-11 所示。

Boehm 在软件质量层次模型中的第三层是软件质量度量。根据软件的需求分析、概要设计、详细设计、实现、组装测试、确认测试和维护与使用七个阶段，制定了针对每一个阶段的问卷表，以此实现软件开发过程的质量控制。

对于企业来说，不管是定制，还是外购软件后的二次开发，了解和监控软件开发过程中每个环节的进展情况、产品水平都是至关重要的，因为软件质量的高低，很大程度上取决于用户的参与程度。

应用 Boehm 模型进行软件质量评价时需要注意：

(1) 对于不同类型的软件(系统软件、控制软件、管理软件、CAD 软件、教育软件、网络软件，以及不同规模的软件，对于质量要求、评价准则、度量问题的侧重点有所不同，应加以区别，如表 7-3 所示。

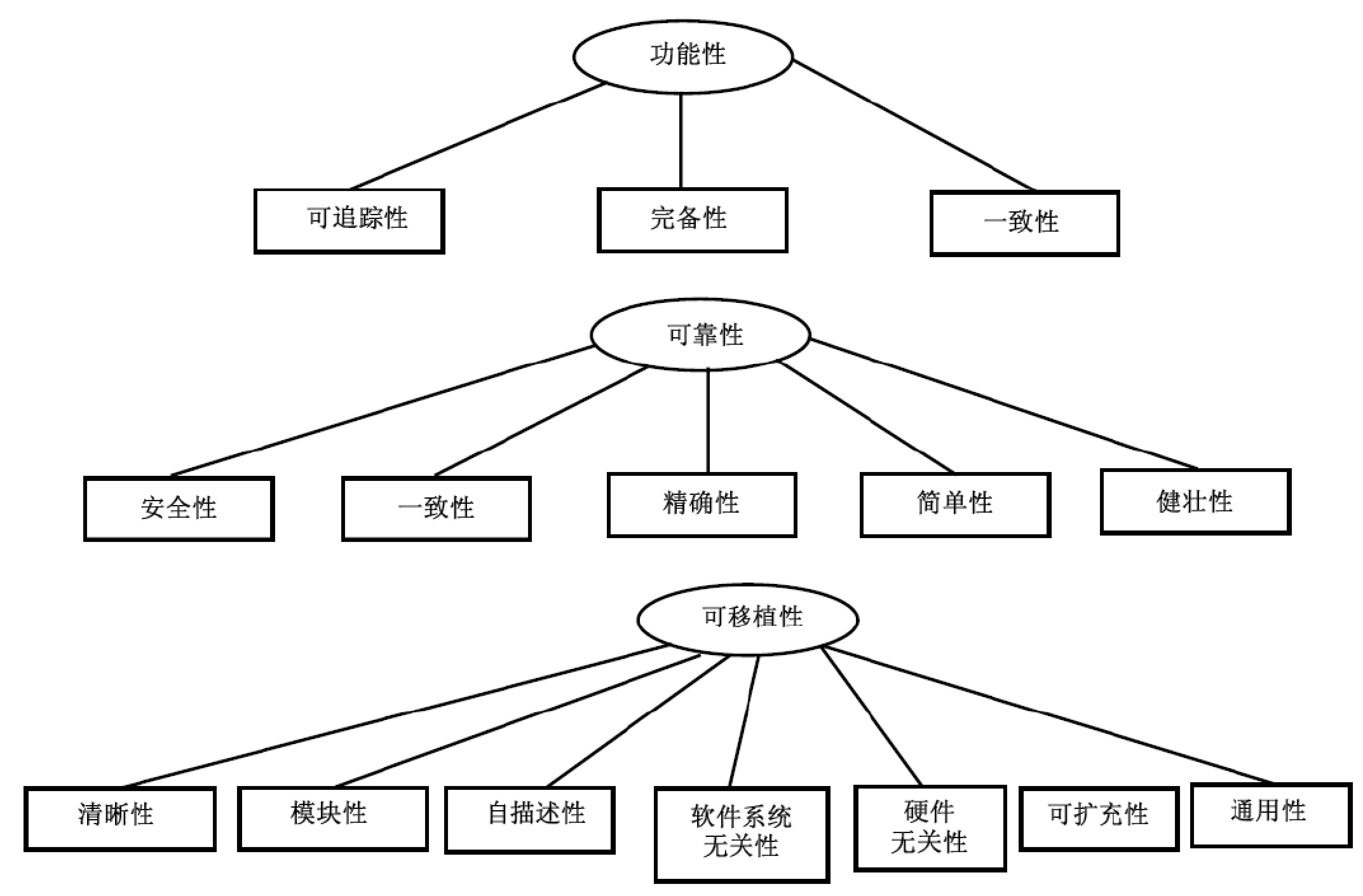


图 7-11　Boehm 质量模型第一层与第二层的关系

表 7-3　软件质量评价的着重点

应用环境特征	需要考虑的要素
生命周期长	可移植性、可维护性
实时系统	可靠性、效率
要在不同的环境中使用	可移植性

(2) 在需求分析、概要设计、详细设计及其实现阶段，主要评价软件需求是否完备，设计是否完全反映了需求以及编码是否简洁、清晰。而且每一个阶段都存在一份特定的度量工作表，它由特定的度量元素组成，根据度量元素的得分就可逐步得到度量准则及质量要素的得分，并在此基础上做出评价。

(3) 对软件各阶段都进行质量度量的根本目的是以此控制软件成本和开发进度，改善软件开发的效率和质量。

3. ISO/IEC 9126 质量模型

ISO/IEC 9126 将软件质量定义为前面所介绍的六大特性：功能性、可靠性、可用性、效率、可维护性和可移植性，每个特性包括一系列子特性。

(1) 软件的功能性主要应该考查三个方面。首先软件产品的功能是否满足需求；其次现有功能是否达到设计要求；最后，所有功能是否实现正常。

(2) 软件的可靠性进一步定义了成熟性(maturity)、容错性(fault tolerance)、易恢复性(recover ability)三个子特性。

(3) 软件的可用性进一步定义了可理解性(understand ability)、易学性(learn ability)、可操作性(operability)三个子特性。

(4) 软件的效率进一步定义了时间特性(time behaviour)和资源特性(resource behaviour)两个子特性。

(5) 软件的可维护性进一步定义了易分析性(analys ability)、易改变性(change ability)、稳定性(stability)、易测试性(test ability)四个子特性。

(6) 软件的可移植性进一步定义了适应性(adapt ability)、易安装性(install ability)、遵循性(conformance)、易替换性(replace ability)四个子特性。

4. 质量模型 GB/T 16260-2006

我国于 2006 年颁布的《信息技术软件产品评价质量特性及其使用指南》GB/T 16260-2006 在 ISO/IEC 9126 质量模型的基础上，对软件质量从 6 个质量特性和 27 个质量子特性进行概念性的描述。图 7-12 给出了质量特性与质量子特性之间的关系。表 7-4 给出了质量特性与质量子特性的描述。

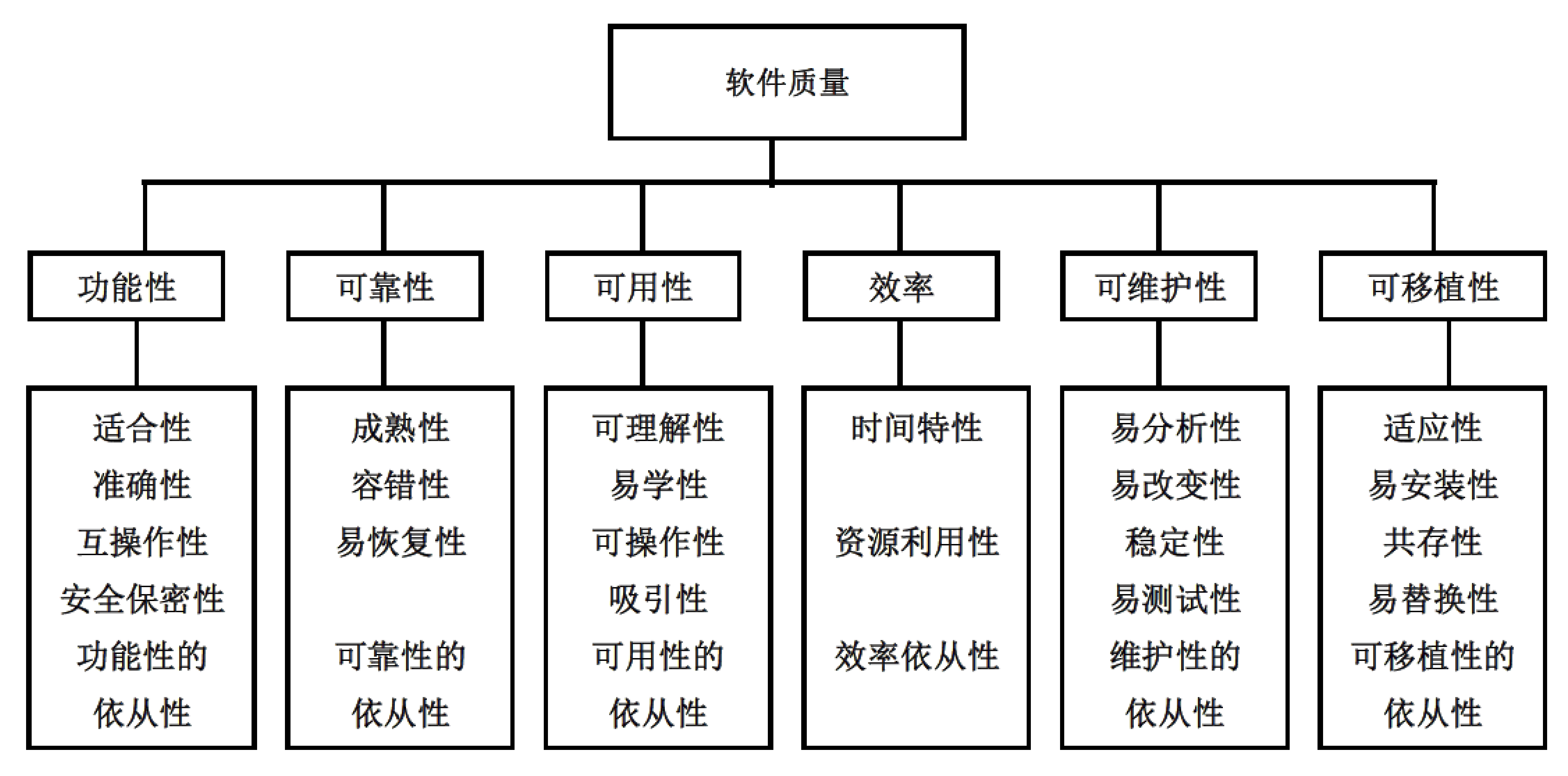


图 7-12　GB/T 16260-2006 质量模型

注意：表 7-4 中的依从性实质上分别对应着功能性的依从性(软件产品遵循与功能性相关的标准、约定或法规以及类似规定的能力)、可靠性的依从性(软件产品遵循与可靠性相关的标准、约定或法规的能力)、可用性的依从性(软件产品遵循与可用性相关的标准、约定、风格指南或法规的能力)、效率依从性(软件产品遵循与效率相关的标准或约定的能力)、维护性的依从性(软件产品遵循与维护性相关的标准或约定的能力)、可移植性的依从性(软件产品遵循与可移植性相关的标准或约定的能力)。

从上我们看出，软件质量特性/子特性之间存在相互冲突，GB/T 16260-2006 的质量模型是面向所有软件的，因此它的质量属性面面俱到。但是对于具体的软件产品或软件项目来说，我们必须考虑利弊，全面权衡，根据质量需求，适当合理地选择/设计质量特性，并进行评价。标准中规定的质量特性、子特性、度量元不一定都要涉及。也就是说，要根据软件产品本身的特点、领域、规模等因素来选择标准中的质量特性、子特性，建立自己的质量模型，其中包括度量元的确定。关于度量元的确定可以从标准中选取，也可以根据实际情况补充若干度量元(因为标准中的度量元不是完备的)，但体系最好与标准一致，即要有名称、度量目的、公式、指标、标度类型等内容。

表 7-4　质量特性与质量子特性的描述

质量特性	描　述	质量子特性	质量子特性描述
功能性	与一组功能及指定的性质有关的一组属性，这里的功能是指满足明确或隐含的需求的那些功能	适合性	与规定任务能否提供一组功能及这组功能的适合程度有关的软件属性
		准确性	与能否得到正确或相符的结果或效果有关的软件属性
		互操作性	与其他指定系统进行交互的能力有关的软件属性
		安全保密性	与防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性
		依从性	是软件遵循有关的软件标准、约定和法规及类似规定的软件属性
可靠性	在规定的一定时间和条件下，与软件维持其性能水平的能力有关的一组属性	成熟性	与由软件故障引起失效的频度有关的软件属性
		容错性	在软件故障或违反指定接口的情况下，与维持规定的性能水平的能力有关的软件属性
		可恢复性	在失效发生后，与重建其性能水平并恢复直接受影响数据的能力，以及为达此目的所需的时间和能力有关的软件属性
可用性	与一组规定或潜在的用户为使用软件所需付出的努力和为这样使用所做的评价有关的一组属性	可理解性	与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性
		易学性	与用户为学习软件应用所花的努力有关的软件属性
		可操作性	与用户为操作和运行控制所花努力有关的软件属性
效率	在规定的条件下，与软件的性能水平与所使用资源量之间关系有关的一组属性	时间特性	与软件执行其功能时响应和处理时间及吞吐量有关的软件属性
		资源特性	与软件执行其功能时所使用的资源数量及使用时间有关的软件属性
可维护性	与进行指定的修改所需付出的努力有关的一组属性	可分析性	与为诊断缺陷或失效原因以及为判定待修改的部分所需努力有关的软件属性
		可修改性	与进行修改、排除错误或适应环境变化所需努力有关的软件属性
		稳定性	与修改所造成的未预料结果的风险有关的软件属性
		可测试性	与确认已修改软件所需付出的努力有关的软件属性
可移植性	与软件可从某一环境转移到另一环境的能力有关的一组属性	适应性	与软件无须采用有别于为软件准备的活动或手段就可能适应不同的规定环境有关的软件属性
		易安装性	与在指定环境下安装软件所需努力有关的软件属性
		一致性	使软件遵循与可移植性有关的标准或约定的软件属性
		可替换性	与软件在软件环境中用来替代指定的其他软件的机会和努力有关的软件属性

7.4.3　软件质量度量与评价

软件产品的质量无法像硬件那样用很多技术指标来衡量，如重量、体积、温度系数等，

但也不能用功能齐全、结构合理、层次分明、执行正确、符合用户规定的功能来概括，因为软件产品仅仅满足这些是远远不够的。另外，在软件项目的开发过程中，往往强调软件必须完成的功能、进度计划、花费成本，而忽略软件工程生命周期中各阶段的质量标准。最后，我们在评价软件质量时应强调软件总体质量(低成本、高质量)，而不应片面强调软件的正确性，忽略其可维护性与可靠性、可用性与效率等；应在软件工程化生产的整个生命周期的各个阶段都注意软件的质量，而不能只在软件最终产品验收时注意质量；应制定软件质量标准，定量地评价软件质量，使软件产品评价走上评测结合、以测为主的科学轨道。

下面从软件生命周期简述各个阶段应该考虑的评价准则，如表 7-5 所示。

作为独立的第三方软件测试组织，对软件进行静态测试，并对软件产品质量进行评测，或对软件产品质量进行度量和评估的依据以及所采用的模型，就是前面叙述的软件质量框架“质量特征—质量子特征—度量元”三层质量模型。

在软件开发中，软件度量的根本目的是管理的需要。没有软件过程的可见度，就无法对软件进行管理，没有软件产品质量的定量描述，就无法对软件质量进行评价。度量是一种可用于决策的可比较的对象。软件度量包含费用、工作量、生产率、性能、可靠性和质量等方面的度量。对于软件质量度量，应根据软件质量要求，确定各个质量特性要求的级别(评定等级)，标识每个质量子特性所要求的度量元和度量方法(事实上，软件质量特性和子特性描述的软件度量需求很难直接测量，需要进一步确定相关的度量元，并将它们与质量子特性、质量特性以及质量模型联系起来)，最终确定软件产品质量的定量定级水平。

表 7-5　软件生命周期各阶段质量的评价准则

开发阶段	评价准则
系统需求分析和设计	完备性、处理有效性、设备有效性、可操作性、培训性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
软件需求分析	完备性、精确性、处理有效性、设备有效性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
设计	精确性、健壮性、处理有效性、完备性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
编码	精确性、健壮性、一致性、可追踪性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
单元测试	精确性、健壮性、处理有效性、可操作性、通信有效性、完备性、一致性、可追踪性、可见性、模块性、清晰性、简单性、结构性
验收	完备性、精确性、健壮性、处理有效性、设备有效性、可操作性、培训性、一致性、可追踪性、文档完备性
维护	精确性、健壮性、设备有效性、可操作性、培训性、一致性、可追踪性、可见性、可扩充性、清晰性、文档完备性

软件产品质量评价准则是用来确定特定软件产品的总体质量是否能够被接受的已经定义成文的规则和条件的集合。

1. 软件质量的度量过程

软件质量度量就是从整体上对软件质量进行测评，用于软件开发中对软件进行质量控制，并最终对软件产品进行评价和验收。

IEEE Std 1061 软件质量度量方法学提供了系统地进行软件质量度量的途径，跨越整个软件生命周期，并包括下列 5 个步骤：

(1) 建立软件质量需求

质量需求表达了在具体应用的特定环境下对软件产品质量的定量要求，应该在软件开发前或初期进行定义，它是有效构造软件质量和客观评价质量的前提。质量需求规格说明可通过所需质量特性的直接度量及其直接度量目标值进行定量表示。直接度量用来验证最终产品是否达到了质量需求。

(2) 准备度量

由软件质量特性和子特性描述的软件质量需求常常无法直接测量，需要进一步确定相关的度量元。在度量的准备阶段，应根据应用环境，为软件开发的各个阶段及其最终产品分别确定适当的度量元，建立度量元、质量子特性、质量特性的映射模型，确定合理的评估准则。

(3) 实现软件质量度量

数据收集过程规定从数据收集点到度量评价的数据流程，确定有关数据的收集条件，给出工具的使用说明及数据存放规程。在全面实施度量前，最好首先在小范围内试验数据收集和度量计算规程，分析其数据量是否一致、度量要求是否确切，尤其要检查主观判断的数据说明和要求是否清晰；检查样板度量过程的费用，修改或完善费用分析；检查所收集到的数据的准确性、度量单位的合适性、所收集到的数据之间的一致性，确认数据样本的随机性、最小样本数、相似性等。

(4) 分析质量度量结果

分析并报告度量结果不仅要做出度量和评估的结论，还要进行度量元的确认，从而确定哪些度量元的确适用于当前软件质量度量活动并可以用于预测软件质量特性值，根据这些度量值和由此计算得到的直接度量的预测值决定被度量对象是否需要做进一步的度量和分析。

(5) 确认软件质量度量

把预测的度量结果与直接度量结果进行比较，以确定预测的度量是否准确地测定了它们的相关质量要素。

2. 度量元选择原则

在对软件质量特性、子特性进行度量时，要对度量元进行适用性选择，选择原则是：

(1) 选择充分体现该领域软件特征的度量元。

(2) 可操作性好、度量数据易获得且获取的代价较小。

(3) 少而精，规模适中。

(4) 子特性、度量元尽量不相关。

(5) 标准符合性要突出。

在选择度量元并进行实际运用时，我们一定要避免走入软件度量的误区，例如：

(1) 目的不明，事后发现度量的内容与管理无关。

(2) 使用度量去评价个人。

(3) 开发人员拒绝执行，认为会否认其工作业绩。

(4) 度量过多，要求广泛收集数据，程序烦琐，不堪重负。

(5) 认为度量结果报告无法引导管理活动。

(6) 管理部门看到可能发生的问题或无成功的结果，而放弃支持度量工作。

(7) 过分强调单个因素的度量。

3. 软件质量评价指标(评价准则)的确定

针对具体软件产品或软件项目实施度量评价时，要确定评价指标。也就是说，衡量软件产品或中间产品的好坏，质量特性、子特性及度量元的合格与否要给出准绳，给出每个特性、子特性的权重。这样一些数据就需要长期积累、总结，也包括专家的评估确定。

因此我们可以看出，选择合适的软件质量指标体系并使其量化是软件测试与评估的关键。评估指标可以分为定性指标和定量指标两种。理论上讲，为了能够科学客观地反映软件的质量特征，应该尽量选择定量指标，但并不是所有的质量特征都可用定量指标进行描述，有时要采用一定的定性指标。这样，我们在选取评估指标时，可按如下原则来进行：

(1) 针对性：不同于一般软件系统，能够反映评估软件的本质特征，具体表现就是功能性与高可靠性。

(2) 可测性：可定量表示，可通过数学计算、平台测试、经验统计等方法得到具体数据。

(3) 简明性：易于被各方理解和接受。

(4) 完备性：选择的指标应覆盖分析目标所涉及的范围。

(5) 客观性：客观反映软件本质特征，不能因人而异。

另外，我们要注意的是选择的评估指标不是越多越好，关键在于指标在评估中所起的作用。评估时指标太多，会增加结果的复杂性，甚至还会影响评估的客观性。指标的确定一般采用自顶向下、逐层分解的方式，并在动态过程中反复综合平衡。

4. 软件质量定量评价公式

对于软件质量的定量评价，国内外在这方面做了很多研究工作，取得了一定的成果。国外著名软件质量度量和评价产品中都给出了相关的计算公式，如Panorama++、Logiscope、McCabe IQ 等。

图 7-13 中是被测程序源代码度量的几张示意图。

下面我们结合有关公司的软件质量定量评价公式进行计算公式的介绍。

(1) 可维护性：可维护性指当系统的功能发生变化和升级时，或当发现错误时容易修改的特性，可维护的软件应该是可理解和可测试的，因为只有这样软件人员才能够容易地确定其影响并验证其变化。

可维护性=0.5*可测试性+0.5*可理解性

(2) 可测试性：可测试性指容易验证软件的正确功能，影响程序可测试性的两个软件特性是结构性和复杂性。为了正确地操作，高度结构化的程序容易把系统部件分成几个独立的测试部分，每部分的复杂性对正确地进行测试所需的测试程序量有影响。

可测试性=0.5*结构性+0.5*McCabe 复杂度

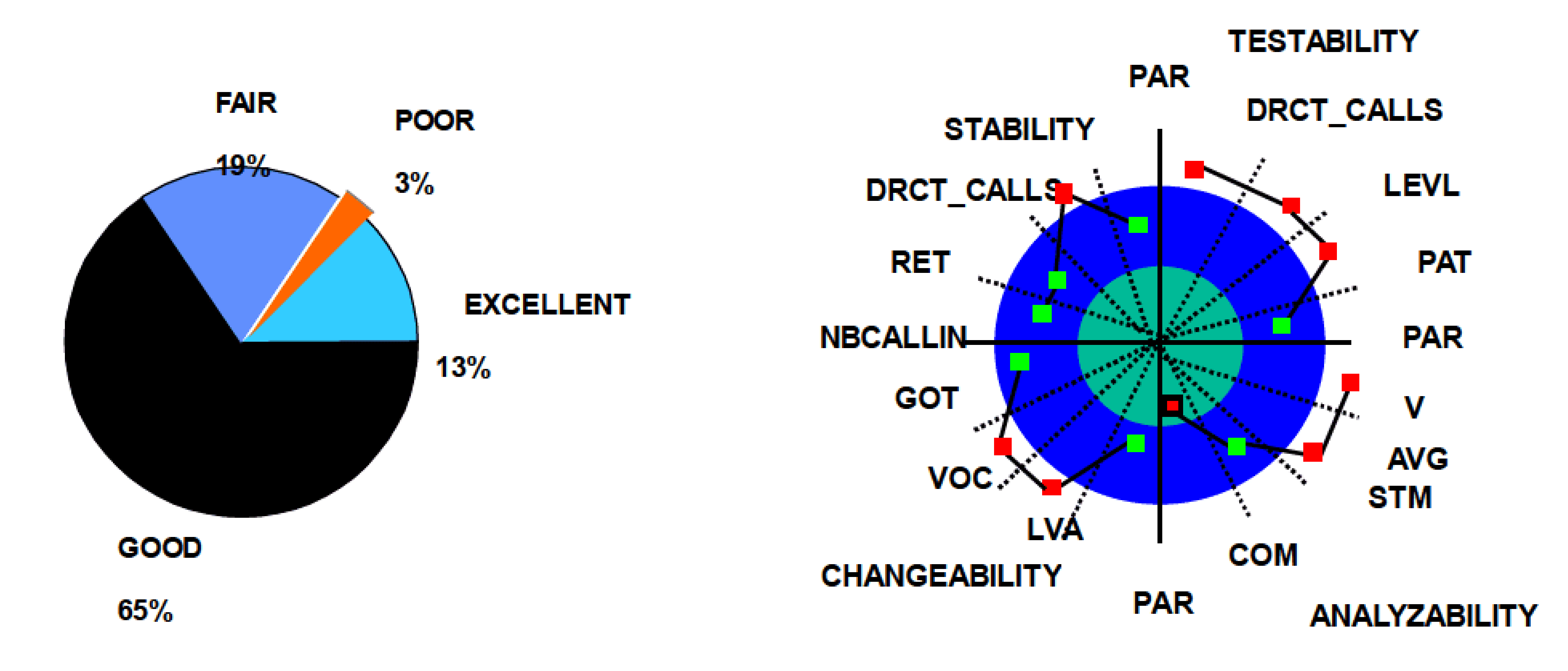


图 7-13　软件质量度量的饼图和 KV 图(雷达图或蜘蛛图)

(3) 可理解性：可理解性指不是原设计者/程序员的那些人员能容易理解程序的功能的程度。它要求简化程序，程序应有描述性注释、良好的结构、最小的复杂性，程序编写要求简明。

可理解性=0.25*结构性+0.25*McCabe 复杂度+0.25*简洁性+0.25*自描述性

(4) 结构性：结构性是指程序自身的特性，它测量程序结构的好坏。衡量程序结构是否良好有下列 5 个方面：①应不修改全局数据；②对逻辑嵌套的深度有所限制；③有单一的返回值，不使用 goto 语句；④使用/设置全部参数类型和对象；⑤推荐使用简单的循环语句而不是 while 循环语句。

结构性=0.2*编码语句的最大嵌套层次+0.2*修改全局数据+0.2*使用 goto 语句+0.2*数据习惯用法+0.2*无条件循环语句所占比例

(5) 复杂性：复杂性反映全部程序及其部件的复杂状态。对于程序单元(过程或函数)，一般采用 McCabe 圈复杂度计算复杂性，分析整个编码的执行控制；对于应用程序，程序单元数量和它们之间的相互关系影响复杂性。

复杂性 =所有模块复杂性/所有模块

模块复杂性=(圈复杂度+模块设计复杂度+设计复杂度+集成复杂度)/(圈复杂度临界值+模块设计复杂度临界值+设计复杂度临界值+集成复杂度临界值)

(6) 简洁性：简洁性指多余信息不在程序中出现的程度。

简洁性=0.4*实体的习惯用法+0.4*局部调用+0.2*被调用

其中，实体的习惯用法={[1−(未使用非输出对象声明/对象声明)]+
[1−(未使用非输出类型声明/类型声明)]+
[1−(未使用非输出参数声明/参数声明)]}/3

局部调用是在同一个封闭的父单元内对其他程序单元的调用。

(7) 自描述性：自描述性衡量程序如何详细地描述自己，自动检查是否存在特殊类型的注释，以判断程序本身描述的质量。

自描述性 = 0.2*注释段 + 0.3*全部注释行所占的比例 + 0.5*注释实体所占比例

(8) 可移植性：可移植性指在一种平台上开发的程序能容易地移植到另一种平台上的程度，使得系统的改变对操作不会有不利的影响。

可移植性 = 0.5*独立性 + 0.5*完整性

(9) 独立性：独立性表示程序与开发环境或主机环境脱离的程度。

独立性 = 0.5*异常比例 + 0.5*用户定义类型

(10) 完整性：完整性指没有信息遗漏，测量程序被完成的程度。

完整性 = (if 语句 + case 语句 + 初始化对象)/3

(11) 可靠性：可靠性指测量程序正确操作的置信度，软件的缺陷越少，可靠性越高。

可靠性 = 0.33*完整性 + 0.33*模块性 + 0.34*可测试性

(12) 模块性：模块性指程序按其功能分割成几个独立的程序单元的程度。用独立的例行程序实现独立功能的那些程序具有高度的模块性。

模块性 = 0.5*编码行数 + 0.5*结构性

一般高级语言的源程序度量都由一些基本的度量元支持，如 McCabe 圈复杂度、注释率、嵌套层数、执行路径个数、宏定义数、函数参数个数、指令个数、goto 数、return 数、扇入/扇出数等；而面向对象语言有关面向对象属性的度量有：每个类的含权方法数、类的可测试性、子孙数、祖先数、继承数深度、对象间耦合、多继承表示、类的扇入/扇出数、类的注释率、类耦合、重定义方法数等。

软件度量过程如图 7-14 所示。

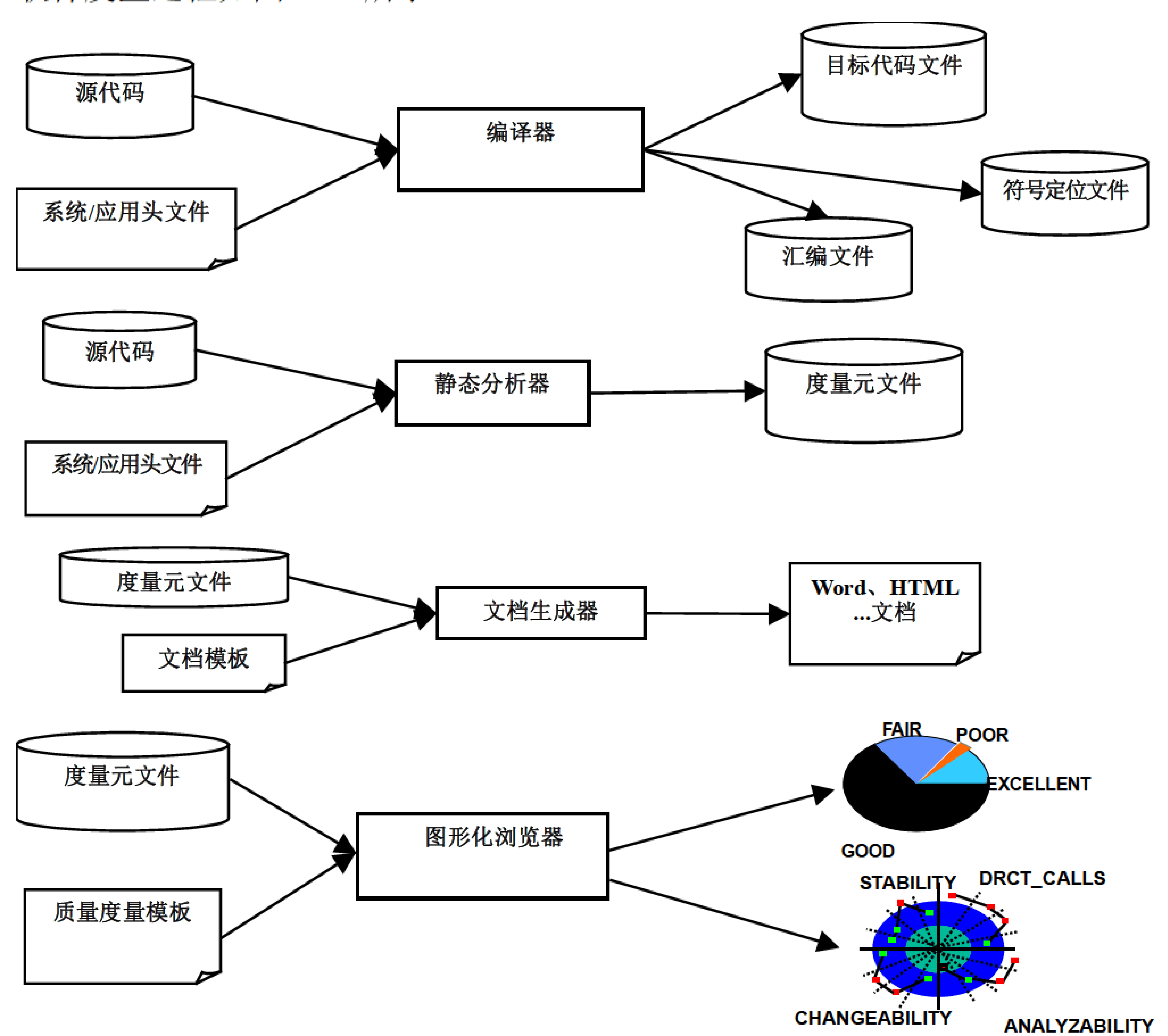


图 7-14　软件度量过程示意图

7.5　代码静态分析工具

代码静态分析主要是进行代码的检查、代码结构的分析、代码问题的查找和代码质量的度量。它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具进行。代码静态分析主要是分析和检查代码与设计的一致性、代码对标准的遵循，以及代码的可读性、代码逻辑表达的正确性、代码结构的合理性等方面；可以发现违背程序编写的标准问题，程序中不安全、不明确和模糊的部分，找出程序中的不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

代码静态分析工具能够帮助人们保证代码的质量，发现并警告代码中潜在的错误。代码静态分析工具和编译器的某些功能其实是很相似的，它们都是利用编译器的前端功能进行词法分析、语法分析、语意分析，并收集各种用于代码分析的结果。代码静态分析工具和编译器的不同之处在于，它们可以自定义各种各样的复杂规则来对代码进行分析，并利用先进的图形表示手段给出各种分析的图形结果。

代码静态分析工具的实现会因为实现方法、算法、分析的层次不同，而在功能上差异很大。总的来说，商用的代码静态分析工具功能齐全、完整，分析结果的可视化效果好，在经济条件许可的情况下是最佳选择；开源的代码静态分析工具尽管功能受限，但在某些方面它们还是有各自优势的。另外，目前有很多 IDE 已经将很多可以用于代码静态分析的功能紧密地集成在了一起，甚至提供了插件的接口来扩展其代码静态分析能力。

7.5.1　编程规则检查工具 CheckStyle

代码规则检查看到的是问题本身而非征兆，能快速找到缺陷，发现 30%~70%逻辑设计和编码缺陷，比动态测试更有效率。作为保障软件质量的重要手段，代码规则检查已成为软件从设计到实装过程中必不可少的一个环节。使用代码规则检查工具来提高测试效率与测试质量是非常有必要的。

CheckStyle 是非常优秀的代码规则检查工具，可以大幅地提高代码质量，当项目的开发人员比较多时，用它来统一代码风格是很有必要的。

它可以根据设置好的编码规则来检查代码，比如符合规范的变量命名、良好的程序风格等。如果项目经理开会时说，“我希望我们写出来的代码就像是一个人写的！”，此时用 CheckStyle 绝对是正确选择。

需要强调的是，CheckStyle 只能做检查，而不能修改代码。想修改代码格式，请使用 Jalopy。它和 CheckStyle 配合使用非常合适。

CheckStyle 的配置性极强，可以只检查一种规则，也可以检查三四十种规则。可以使用 CheckStyle 自带的规则，也可以自己增加检查规则。它支持几乎所有主流 IDE，包括 Eclipse、IntelliJ、NetBeans、JBuilder 等。Eclipse 的 CheckStyle 插件的下载地址为 http://sourceforge.jp/projects/sfnet_eclipse-cs/releases/。

1. 插件的安装

首先这个插件不是 Eclipse 自带的，而需要去网上下载。得到插件之后，在 Eclipse 的 links 文件夹中新建一个名为 checkstyle_4.2.link 的文件，用记事本打开，在里面设置 path 来记录插件的安装路径：path=D://tool//checkstyle_4.2。注意路径用“//”隔开，否则不会显示出插件的内容。在 Eclipse 的 Window|Preferences 中可以看到 Checkclipse 的选项，如图 7-15 所示。

Checkclipse有两个渠道可以进行配置，一个是全局的，另一个是单个项目(Project)的。全局的可以在整个Eclipse的workbench中生效，而单个项目的配置可以在指定的项目中生效，它优先于全局的配置。

对于单个项目：右键单击某个项目，然后选择 Properties 命令就可以看到 Checkclipse 的窗口。在 Configuration 选项卡中，选中 Enable Checkstyle 复选框，然后在 Checkstyle Configuration File 中选择 CheckStyle 配置文件就可以了，如图 7-16 所示。

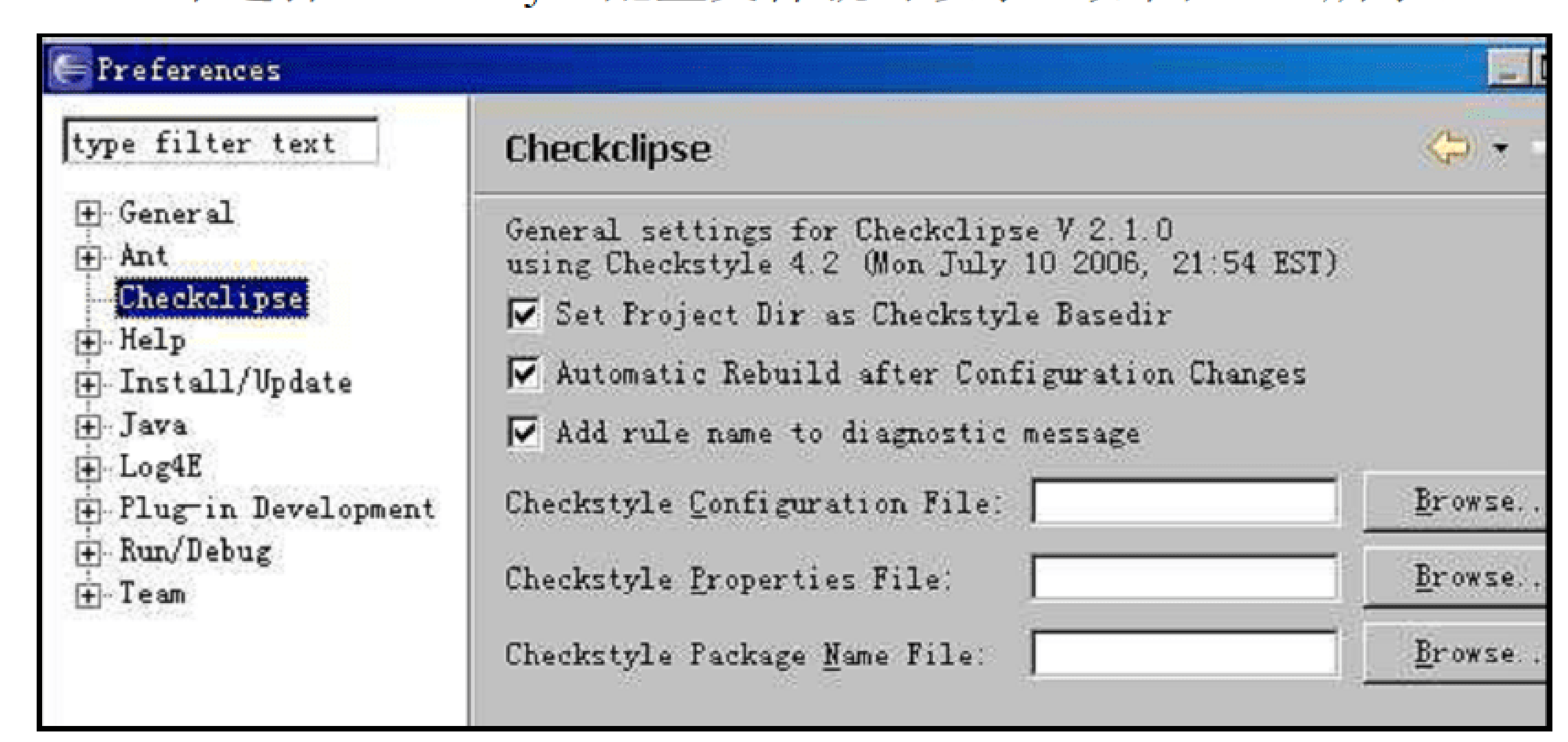


图 7-15　成功安装 Checkclipse 后的 Preferences 窗口

对于全局的配置：选择 Window|Preferences 命令就可以看到。设置方法与单个项目的设置是一样的。

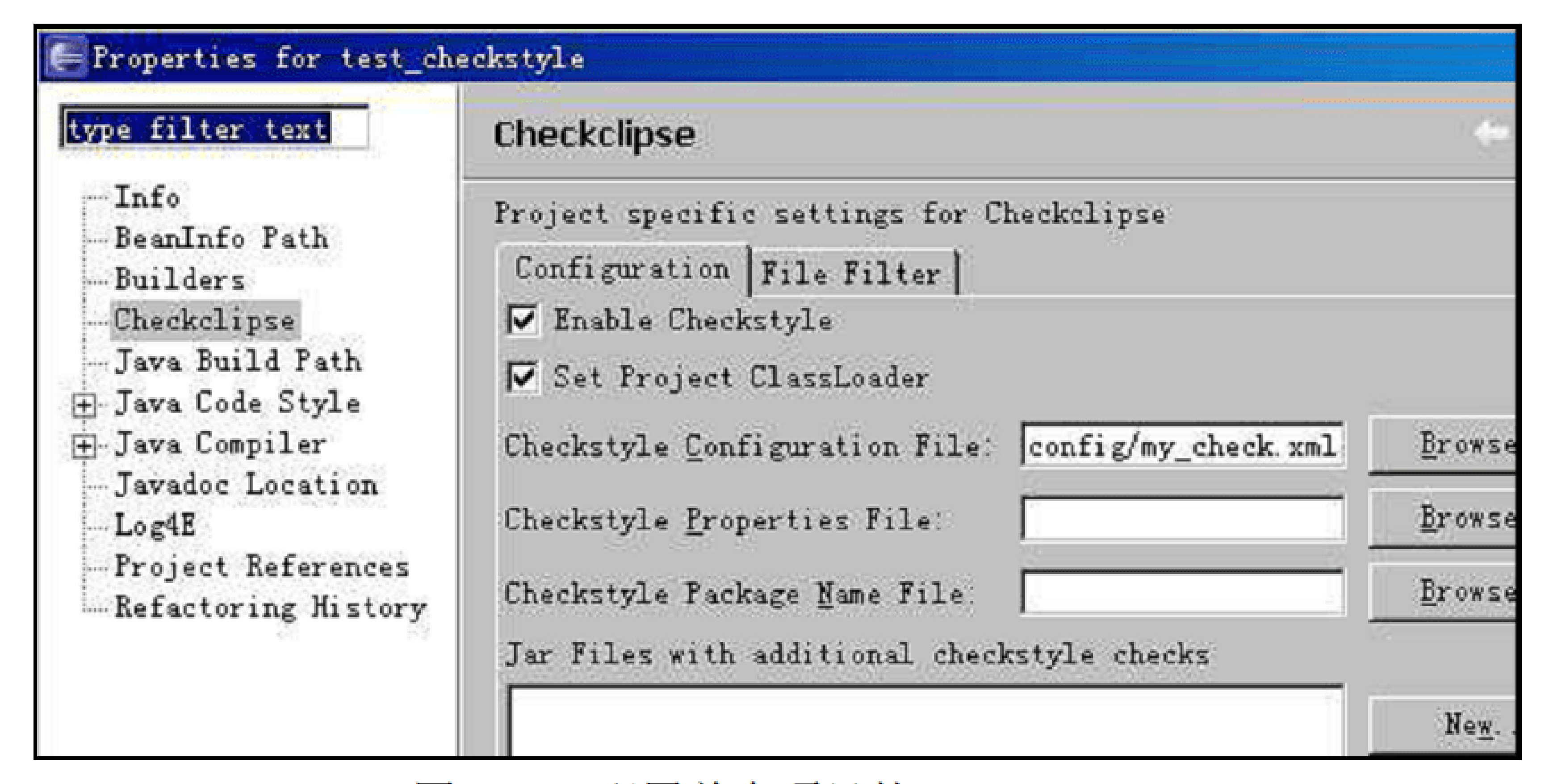


图 7-16　配置单个项目的 Checkclipse

经过上面的设置，Checkclipse 就可以使用了。如果想设置需要被检查的文件名，那么就在 File Filter 标签中修改被包含的文件。可以使用 Add、Remove、Change 等按钮进行编辑。Included Resources 中显示了被检查的文件清单，如图 7-17 所示。

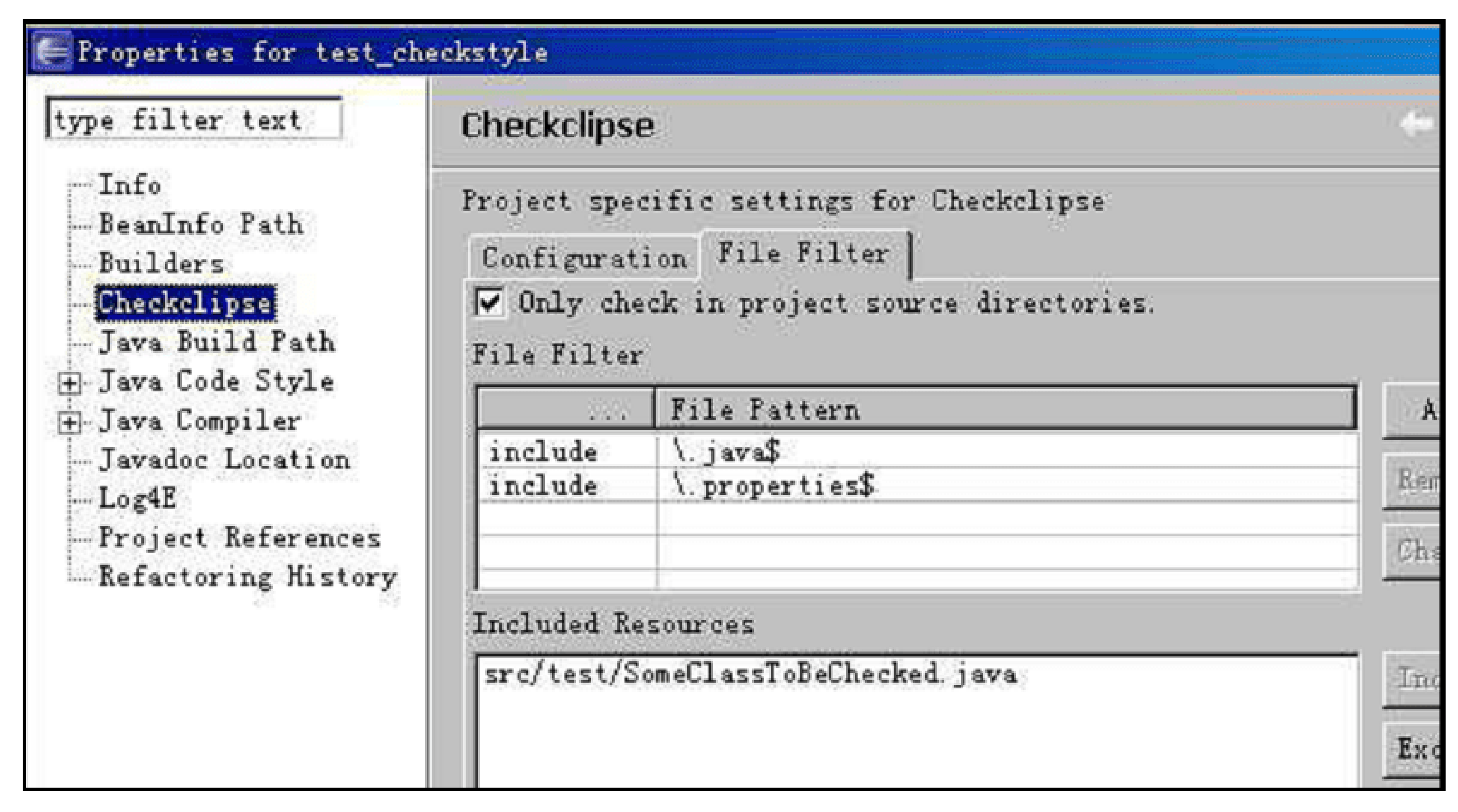


图 7-17 设置单个项目的 Checkclipse 的文件过滤器(file filter)

2. Checkclipse 的使用

(1) 建立一个 Eclipse 项目 test_checkstyle，里面包含一个源文件夹 src，一个目标生成文件夹 eclipse_build，如图 7-18 所示。

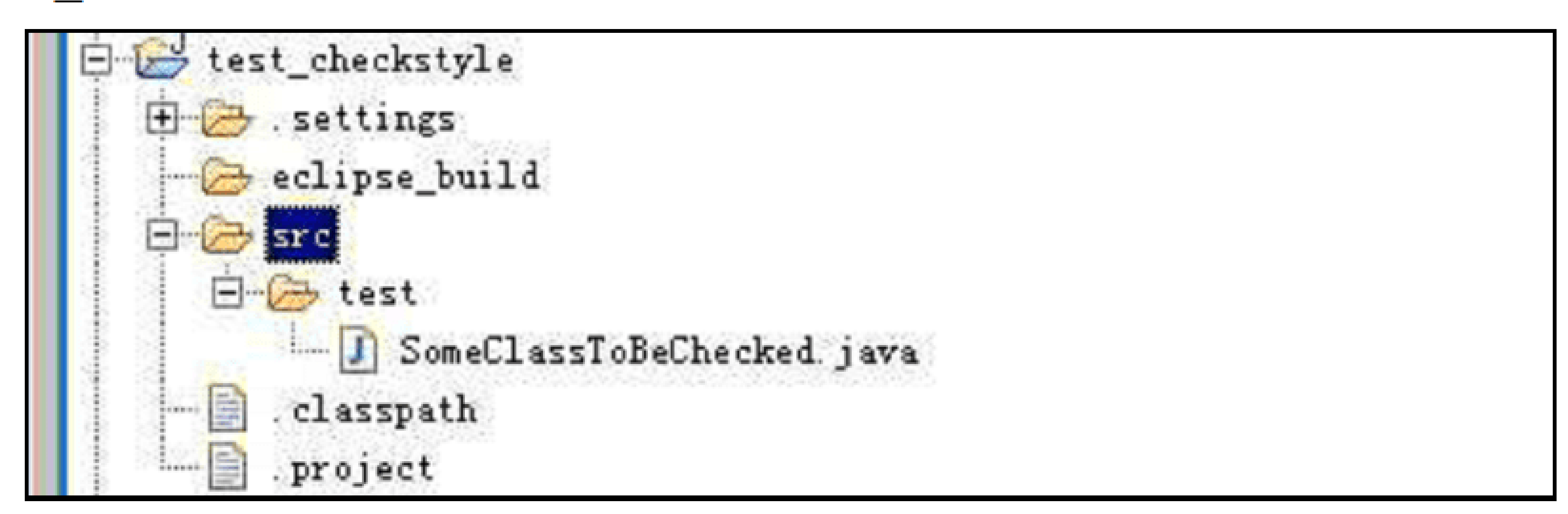


图 7-18 测试如何使用 CheckStyle 中的项目

(2) 在项目中开启 CheckStyle：打开该项目的属性，单击左侧的 Checkclipse 后，选中 Enable Checkstyle 复选框。

(3) 建立一个测试用的类，比如 SomeClassToBeChecked，内容如下。

```
/* Copyright (c) 2001-2008 Beijing BidLink Info-Tech Co., Ltd.
* All rights reserved.
* Created on 2008-2-22
* $Id: learn_in_5_min.xml,v 1.3 2008/03/03 03:43:44 Administrator Exp $*/
package test;
public class SomeClassToBeChecked {
}
```

(4) 用 CheckStyle 检查它：右击项目名，选择 Build Project 命令，对 src 文件夹进行编译，把类文件放到 eclipse_build 中。结束之后，可以看到图 7-19 中代码的第 10 行有一个大括号，把鼠标移上去就会出现提示“Missing a Javadoc comment”。同时，在 Problems 窗口中也有提示，如图 7-20 所示。

```
SomeClassToBeChec...  ×   messages.xml   learn_in_5_min.xml
/*
 * Copyright (c) 2001-2008 Beijing BidLink Info-Te
 * All rights reserved.
 * Created on 2008-2-22
 *
 * $Id$
 */
package test;

public class SomeClassToBeChecked {

}
```

图 7-19　代码窗口中的错误提示

Search	Console	History	Problems ×	
0 errors, 1 warning, 0 infos				
Description	Resource	Path	Loca	
⊟ Warnings (1 item)				
Missing a Javadoc comment.	SomeClassToBeCheck...	test_...	line	

图 7-20　Problems 窗口中的错误提示

(5) 修改代码：既然提示缺少了 Javadoc 注释，就把它加上，如图 7-21 所示。

```
SomeClassToBeChec...  ×   messages.xml   learn_in_5_min.xml
/*
 * Copyright (c) 2001-2008 Beijing BidLink Info-Te
 * All rights reserved.
 * Created on 2008-2-22
 *
 * $Id$
 */
package test;
/**
 * 这是个测试类，为了测试如何使用CheckStyle而建立的。
 * @author 申思维 shensiwei@sina.com
 *
 */
public class SomeClassToBeChecked {

}
```

图 7-21　增加了 Javadoc 注释后的效果

然后重新编译，可以看出，警告没有了，检查通过。

由于 CheckStyle 自带的规则检查非常严格，一般的项目警告非常多。通常可针对需要，定义自己的规则检查。

3. 自定义规则

CheckStyle 没有图形化的定制器，所以需要手工修改配置文件，比如代码需要符合下列规则：

(1) 长度方面：文件长度不超过 1500 行，每行不超过 120 个字，方法不超过 60 行。

(2) 命名方面：类名不能以小写字母开头，方法名不能以大写字母开头，常量不能有小写字母。

(3) 编码方面：不能用魔法数(Magic Number)，if 最多嵌套三层。

检查配置文件(如命名成 my_check.xml)，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC
 "-//Puppy Crawl//DTD Check Configuration 1.2//EN"
 "http://www.puppycrawl.com/dtds/configuration_1_2.dtd">
<module name="Checker">
    <module name="TreeWalker">
 <!-- 长度方面的检查 -->
 <!-- 文件长度不超过 1500 行 -->
 <module name="FileLength">
     <property name="max" value="1500"/>
 </module>
 <!-- 每行不超过 120 个字-->
 <module name="LineLength">
     <property name="max" value="120"/>
 </module>
 <!-- 方法不超过 60 行 -->
 <module name="MethodLength">
     <property name="tokens" value="METHOD_DEF"/>
     <property name="max" value="60"/>
 </module>

 <!-- 命名方面的检查，它们都使用了 CheckStyle 默认的规则。 -->
 <!-- 类名(class 或 interface)的检查 -->
 <module name="TypeName"/>
 <!-- 方法名的检查 -->
 <module name="MethodName"/>
 <!-- 常量名的检查 -->
 <module name="ConstantName"/>
 <!-- 编码方面的检查 -->
 <!-- 不能用魔法数 -->
 <module name="MagicNumber"/>
 <!-- if 最多嵌套三层 -->
 <module name="NestedIfDepth">
     <property name="max" value="3"/>
 </module>
    </module>
</module>
```

可以看出，想增加一个检查，就是增加一个<module/>节点，然后具体写明节点内容。

让 CheckStyle 使用指定的检查配置文件：打开项目属性，在 Checkclipse 中的 CheckStyle Configuration File 一栏中选定配置文件，然后确定，参见前面的图 7-16。

然后重新编译项目，就会发现，CheckStyle 的规则正如我们所愿：只检查在文件中配置的几项，并且它们是以 Error 级别进行提示，而不是默认检查时出现的 Warning 级别。比如在一个方法中增加4层嵌套(共5个 if)，并将方法名大写，就会出现如图7-22所示的结果。

可以看到，出现了两个 Error：方法名的“Name 'xx' must match pattern...”和 if 嵌套的“Nested if-else depth is 4...”。把它们都改过来，即把方法名小写，让 if 循环嵌套三层，然后重新编译即可。

```
*/
package test;

public class SomeClassToBeChecked {
    //类名前无javadoc，但是由于我们没有配置javadoc
    //的检查，所以没报错。

    public void TestNestedIf(){
    //该方法名被大写。所以报错
        if(true){
            if(true){
                if(true){
                    if(true){
                        //嵌套的第四个if，报错
                        if(true){    }
        }}}}
    }
}
```

Search | Console | History | Problems

2 errors, 0 warnings, 0 infos

Description	Resource	Path
Errors (2 items)		
Name 'TestNestedIf' must match p	SomeClassToBeCheck...	test_...
Nested if-else depth is 4 (max a	SomeClassToBeCheck...	test_...

图 7-22　定制配置的检查结果

7.5.2　代码缺陷分析工具 PMD

错误发现得越晚，修正的成本就越高，在测试阶段修正错误的成本约是编码阶段的 4 倍。为了减少成本，错误被发现得越早越好。在编程阶段，通过静态分析找出代码中大部分的错误，是很多人的梦想。这个梦想在 21 世纪初变成了现实。目前 IT 业已经在大量使用代码静态分析工具，以便在编码阶段就能够找出可能的编码缺陷。以 KlocWork 公司的 K7、Coverity 公司的 Prevent、Parasoft 公司的 Insure++、Fortify Software 公司的 SCA 和 Gimpel Software 公司的 PC-Lint 等为代表的商用静态分析软件，以及以 Prefast、Cppcheck、Findbugs、PMD、Splint 等为代表的开源静态分析工具，实现了只要静态分析代码，就可以发现代码中的错误，例如数组越界、除数为 0、缓冲区溢出等。尽管它们或多或少地存在一些问题，还不是特别完美，但对人们的帮助还是相当大的。

1. PMD 功能介绍

PMD 是由 DARPA 在 SourceForge 上发布的开源 Java 代码静态分析工具。最初，PMD 是为了支持 Cougaar 项目而开发的。Cougaar 是美国国防高级研究计划局(Defense Advanced Research Projects Agency，DARPA)的一个项目。PMD 通过其内置的编码规则对 Java 代码进行静态检查，主要包括对潜在的 bug、未使用的代码、重复的代码、循环体创建新对象等问题的检验。PMD 提供了和多种 Java IDE 的集成，例如 Eclipse、IDEA、NetBean 等。

PMD 是一种开源分析 Java 代码错误的工具。与其他分析工具不同的是，PMD 通过静态分析获知代码错误。也就是说，在不运行 Java 程序的情况下报告错误。PMD 附带了许多可以直接使用的规则，利用这些规则可以找出 Java 源程序的许多问题，例如以下这些错误：

(1) 潜在的 bug：空的 try/catch/finally/switch 语句。

(2) 未使用的代码：未使用的局部变量、参数、私有方法等。

(3) 可选的代码：String/StringBuffer 的滥用。

(4) 复杂的表达式：非必需的 if 语句、可以使用 while 循环完成的 for 循环。

(5) 重复的代码：复制/粘贴代码意味着复制/粘贴 bug。

(6) 循环体创建新对象：尽量不要复制 for 或 while 循环体内实例化一个新对象。

(7) 资源关闭：Connect、Result、Statement 等使用之后确保关闭掉。

此外，用户还可以自己定义规则，检查 Java 代码是否符合某些特定的编码规范。例如，可以编写一个规则，要求 PMD 找出所有创建 Thread 和 Socket 对象的操作。

1) 实现原理

PMD 的核心是 JavaCC 语法分析生成器。PMD 结合运用 JavaCC 和 EBNF(Extended Backus-Naur Formal，扩展巴科斯-诺尔范式)文法产生一个分析器，用来分析 Java 源代码(文本)。又在 JavaCC 的基础上加入了语义的概念，也就是 JJTree，这样就把 Java Source 转换成了一个抽象语法树(Abstract Syntax Tree，AST)，AST 是一个结构化的对象层次结构。可以用访问者模式访问这个结构上的每个节点，从而找出哪个节点违反了哪些规则。

2) 实现过程

首先传一个文件名或 Ruleset 给 PMD，PMD 把该文件流传给自己生成的 JavaCC 分析器，分析完毕后，PMD 就获得了分析生成的 AST 的一个引用。PMD 把 AST 处理成一个符号表，用户可以在该符号表里查询一些有用的信息。每条 PMD 规则都会遍历整个 AST 并检验是否发生了错误，接着 PMD 产生一个报表，上面说明了有哪些地方违反了 PMD 规则。

3) PMD 规则

(1) PMD 默认规则

PMD 自带了很多规则，并且分类写入不同的 ruleset 文件。

(2) PMD 自定义规则

PMD 自带了很多代码规范的规则，还可以自定义规则，可以把这些规则整合到一起。最后，运行 PMD 的时候就可以指定这个 ruleset 文件，按照需求进行代码检查。

4) 支持的 Java 编辑器

PMD 支持的编辑器包括：JDeveloper、Eclipse、JEdit、JBuilder、BlueJ、CodeGuide、NetBeans/Sun Java Studio Enterprise/Creator、Intelli J IDEA、TextPad、Maven、Ant、Gel、JCreator 和 Emacs。不同编辑器对应的 PMD 插件版本可从网址 http://sourceforge.net/projects/pmd/files/获取。

5) 发版计划

PMD 计划每一两个月发布一个正式版本。2013 年 5 月 1 日发布了最新的 PMD 5.0.4 版，同时在 2013 年 5 月 10 日发布了 PMD for Eclipse 4.0.0.v20130510-1000 版。

6) PMD 源码获取

(1) 通过官网获取源代码：http://sourceforge.net/projects/pmd。

(2) 通过 Git Hub 获取源代码：https://github.com/pmd。

2. PMD 环境建立

可以从 PMD 的网站下载 PMD 的二进制版本，或下载带源代码的版本，下载得到的都是 ZIP 文件。PMD 源码可通过下载地址http://sourceforge.net/projects/pmd/files/pmd/获取。

如果要在一个 Java 源代码目录中运行 PMD，只需要直接在命令行上运行下面的命令：

```
G:\pmd-bin-5.0.4\bin>java –jar ..\lib\pmd-5.0.4.jar G:\cellstyle text rulesets/unusedcode.xml
```

一些可选参数如下。

-debug：打印 debug 日志信息。

-targetjdk：指定目标源代码的版本——1.3、1.4、1.5、1.6 或 1.7；默认是 1.5。

-cpus：指定创建的线程数。

-encoding：指定 PMD 检查的代码的编码方式。

-excludemarker：指定 PMD 需要忽略的行的标记，默认为 NOPMD。

-shortnames：在报告中显示缩短的文件名。

-linkprefix：HTML 源文件的路径，只是为了 HTML 显示。

-lineprefix：自定义的锚，用于影响源文件中的行，只是用于 HTML 显示。

-minimumpriority：规则的优先级限制，低于优先级的规则将不被使用。

-nojava：不检查 Java 文件，默认检查 Java 文件。

-jsp：检查 JSP/JSF 文件，默认不检查。

-reportfile：将报告输出到文件，默认打印到控制台。

-benchmark：输出一个基准清单，默认输出到控制台。

-xslt：覆盖默认的 xslt。

-auxclasspath：指定源代码文件使用的类路径。

在 Eclipse 中安装 PMD 插件的方法有如下两种：

1) 通过 Eclipse 更新安装

PMD 可作为插件集成到很多流行的 IDE 中，很多的插件中都包含 PMD 的 jar 文件，这个 jar 文件中包含规则集。所以虽然一些插件中使用 rulesets/unusedcode.xml 作为参数引用规则集，但是实际上是使用 getResourceAsStream 方法来从 PMD 的 jar 文件中加载。

由于 Eclipse 是比较流行的开源 Java/J2EE 开发 IDE，因此本书主要介绍如何在 Eclipse 中使用 PMD 工具进行代码的检查，并且选用最新的 JDK 1.7 + Eclipse 4.2。在 Eclipse 中安装 PMD 的过程如图 7-23 所示。

最后重启 Eclipse，PMD 即安装成功。

2) 本地安装

下载最新的 ZIP 文件包，然后执行上述过程，只是在 Add Repository 时，选择 Archive 来代替 Update Site，并使用下载的 ZIP 文件。

在安装完更新后，如果发生了一个异常，例如“java.lang.RuntimeException：Could not find that class xxxx”，这时可试着删除 workspace 中的.metadata/plugins/net.sourceforge.pmd.eclipse 目录下的 ruleset.xml 文件。

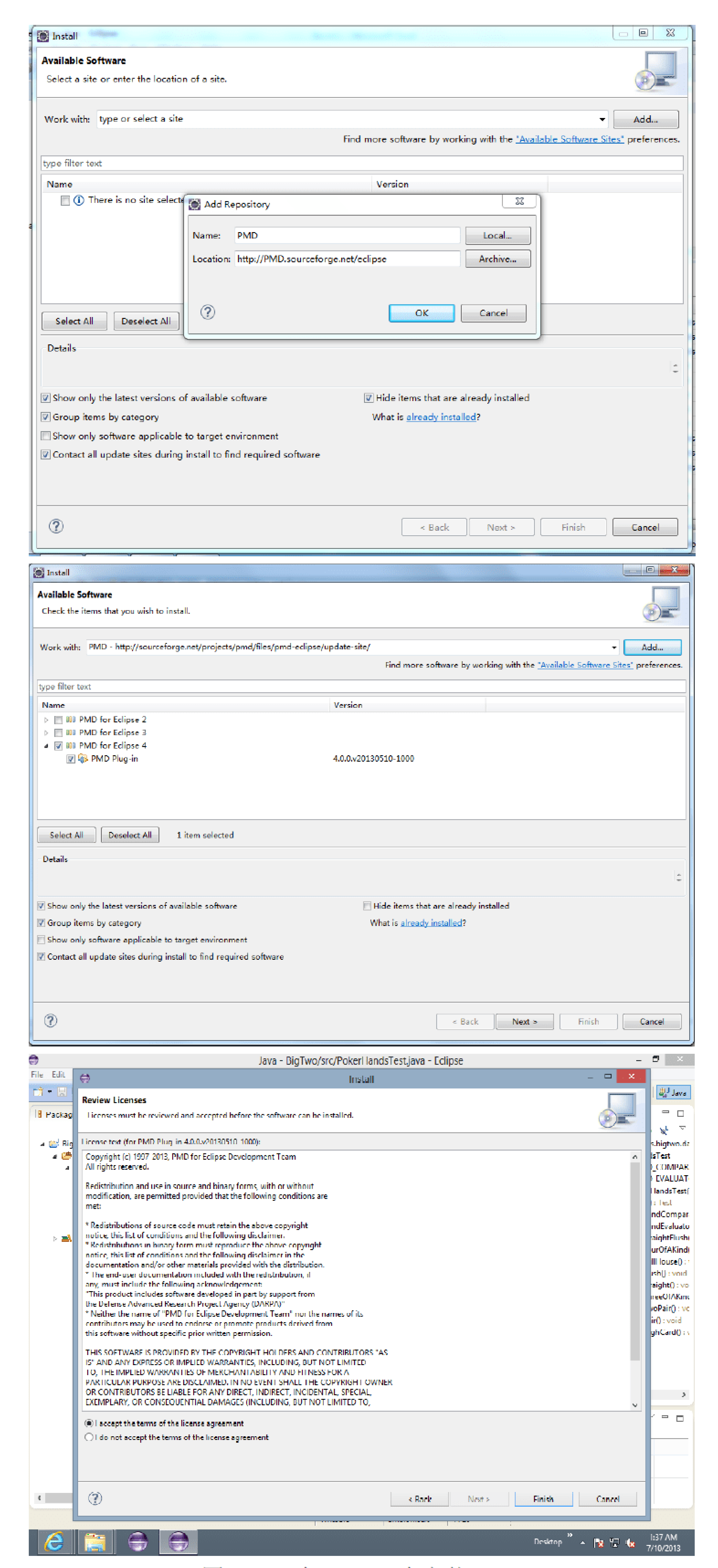


图 7-23　在 Eclipse 中安装 PMD

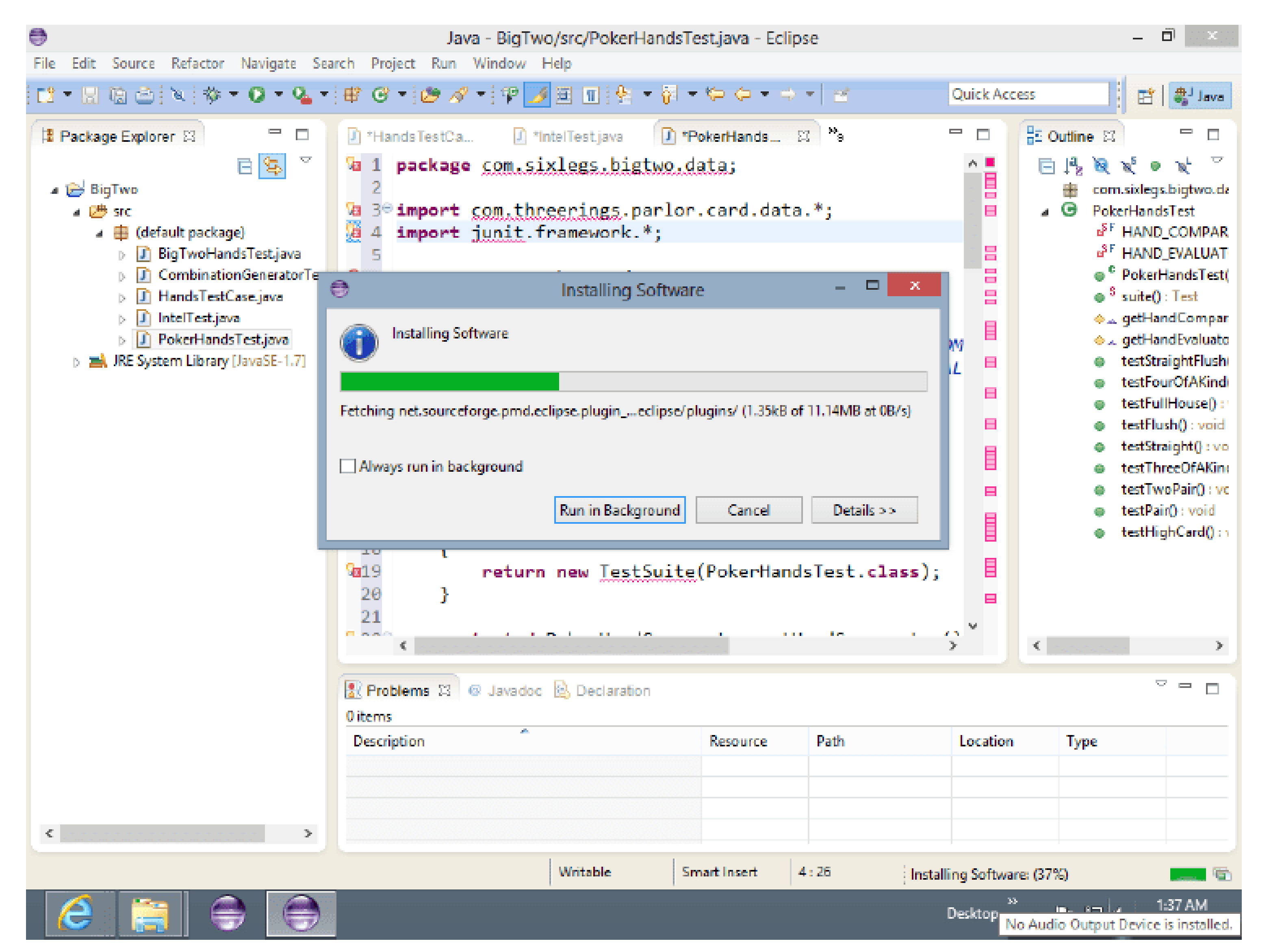


图 7-23(续)

以下为主要的 Ant 配置信息：

```
<path id="pmd.path">
        <filesetdir="${lib.dir}/pmd-5.0.4">
                <include name="**/*.jar" />
        </fileset>
</path>
<taskdefname="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
        classpathref="pmd.path" />
<taskdefname="cpd" classname="net.sourceforge.pmd.cpd.CPDTask"
        classpathref="pmd.path" />
<target name="pmd">
        <pmdshortFilenames="true">
                <ruleset>rulesets/favorites.xml</ruleset>
                <formatter type="html" toFile="d:\foo.html" toConsole="false" />
                <filesetdir="${src.dir}">
                        <include name="**/*.java" />
                </fileset>
        </pmd>
</target>
<target name="cpd">
        <cpdminimumTokenCount="100" outputFile="d:/cpd.txt">
                <filesetdir="${src.dir}">
                        <includename="**/*.java"/>
                </fileset>
        </cpd>
</target>
```

用 Ant 命令运行 build.xml，PMD 就会按照设定好的规则自动执行代码检查。

3. PMD 应用流程

1) PMD 应用流程图(如图 7-24 所示)

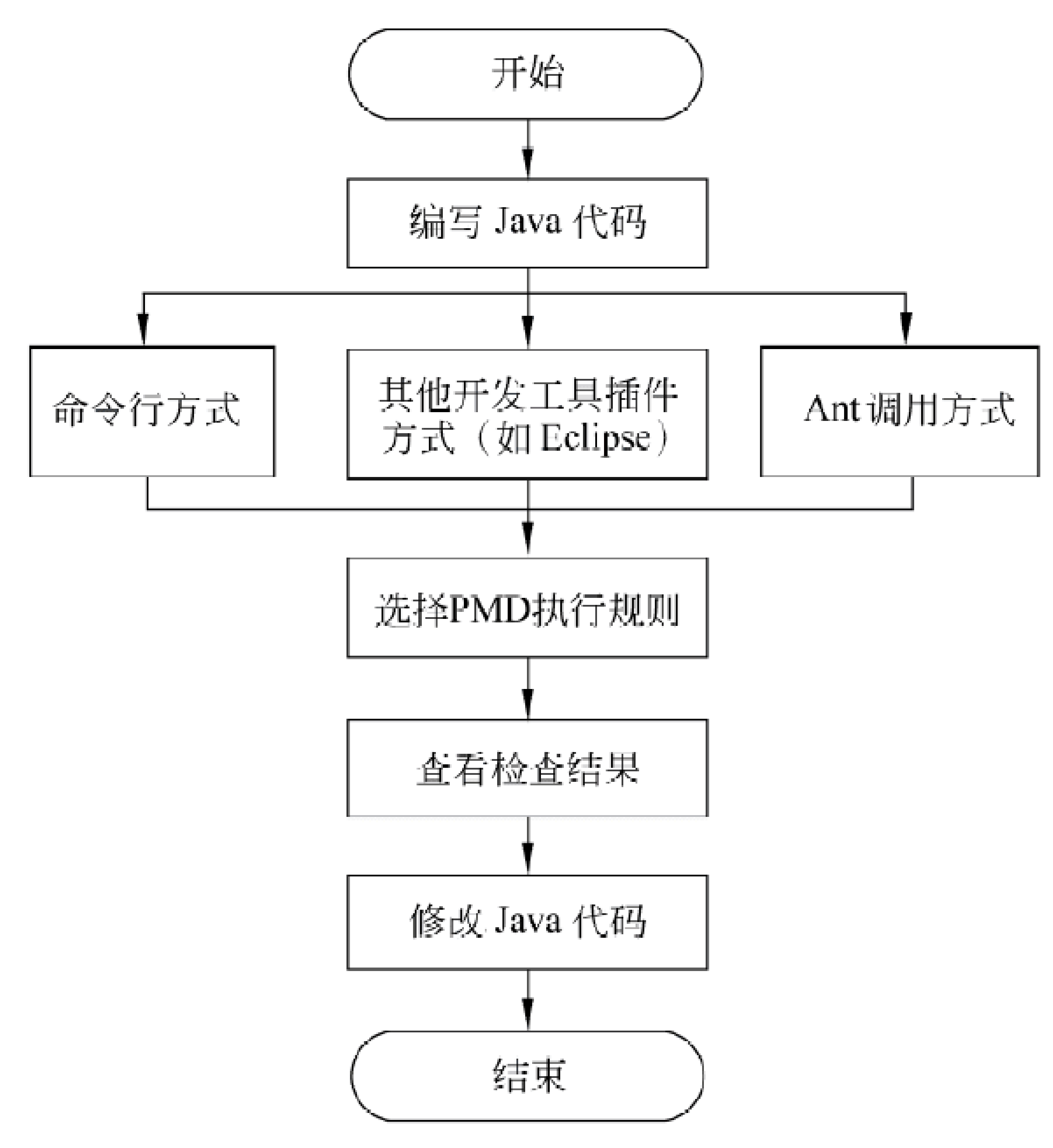


图 7-24　PMD 应用流程图

2) PMD 的使用流程

PMD 的使用流程概括起来可以分为以下几步：

(1) PMD 的执行参数设置

启动 Eclipse IDE，打开工程，选择 Windows|Preferences 下的 PMD 项，可以对 PMD 的一些执行参数进行设置，如图 7-25 所示。

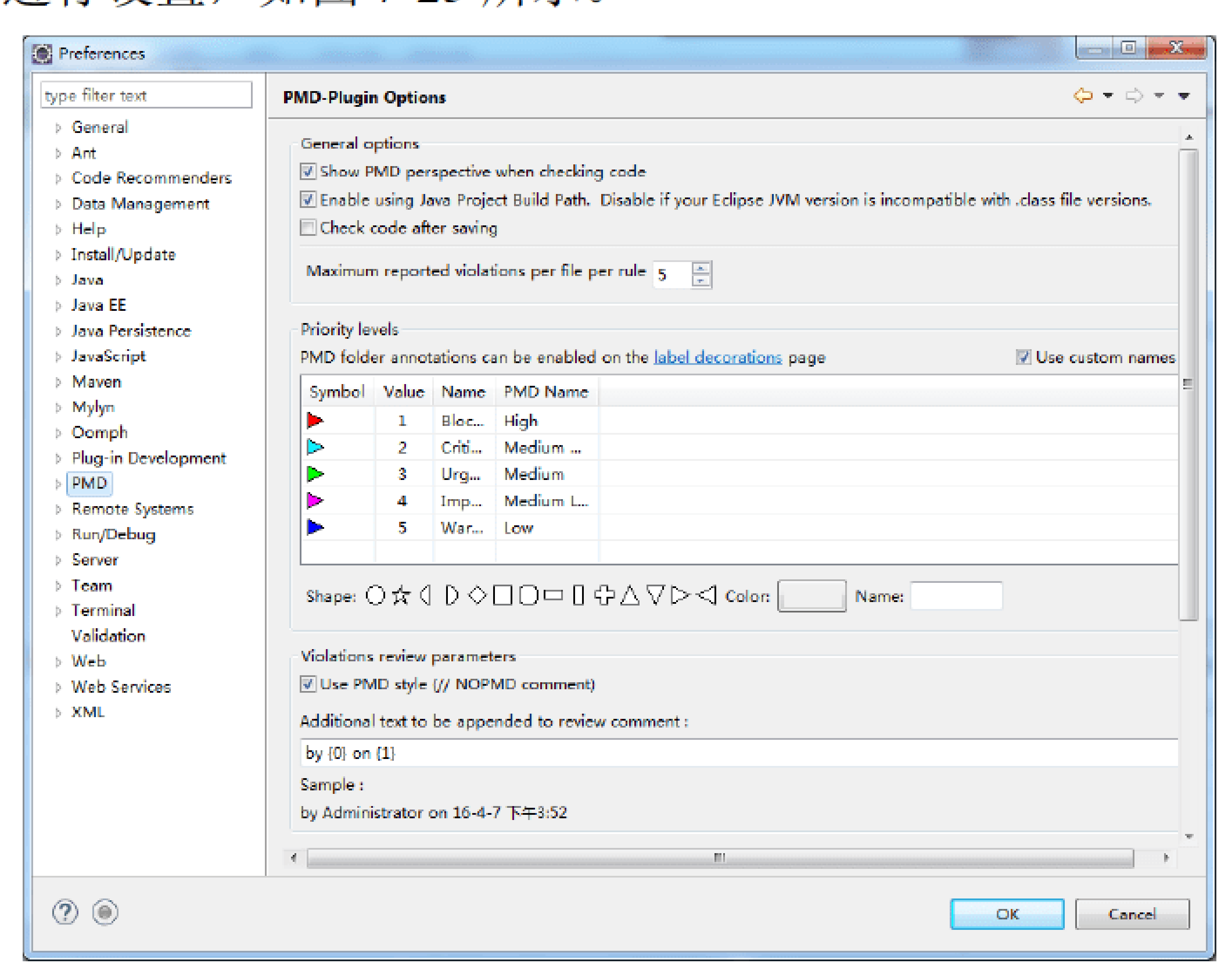


图 7-25　设置 PMD 执行参数

(2) 配置 PMD 的检查规则

通过图 7-25 中左侧 PMD 节点下的 Rules Configuration 节点可以配置 PMD 的检查规则，自定义检查规则也可以在此通过 Import 的方式导入到 PMD 中，如图 7-26 所示。

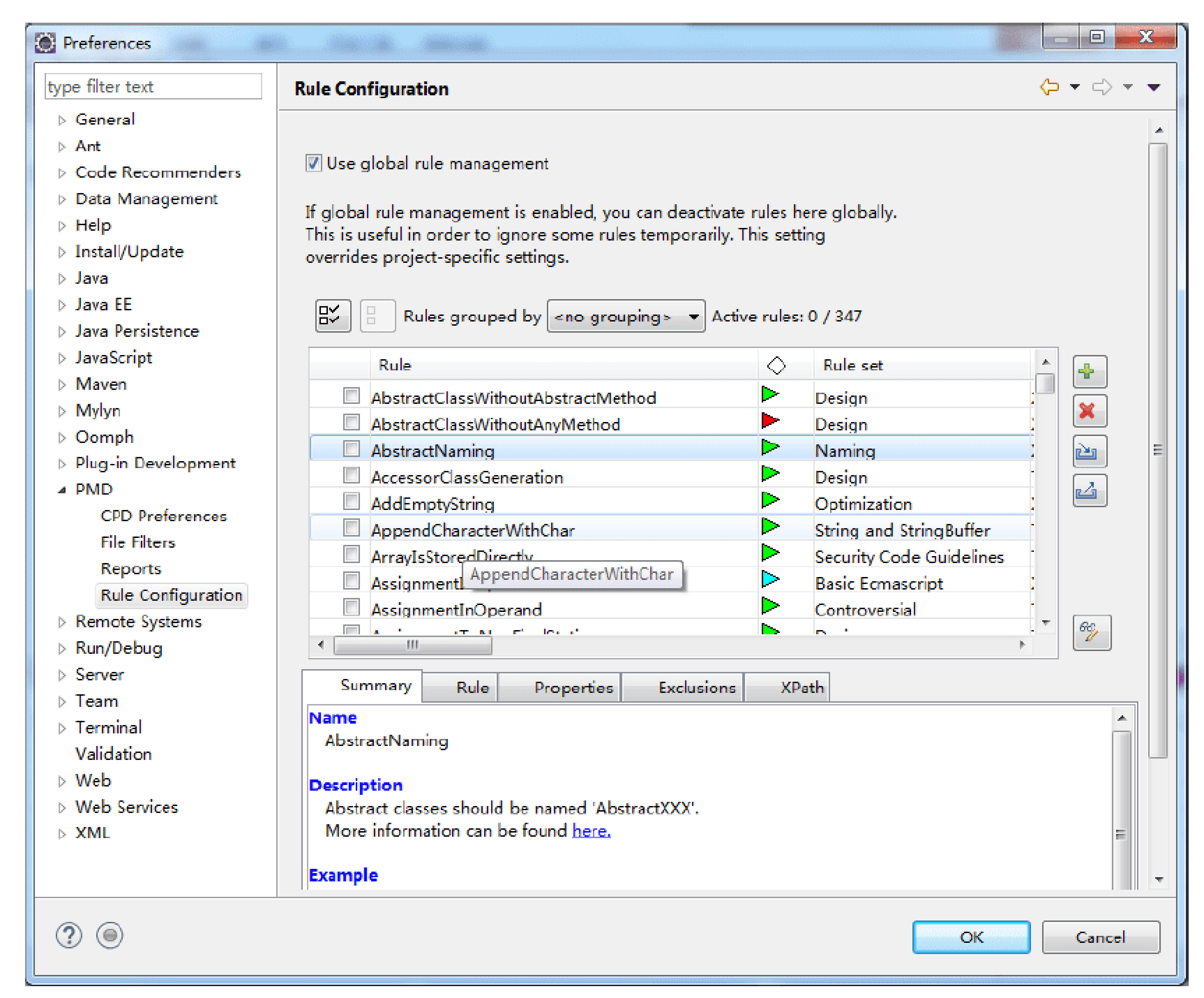


图 7-26　检查规则配置

(3) 检查 Java 程序代码

配置好之后，右击工程中需要检查的 Java Source，选择 PMD|Check Code With PMD，之后 PMD 就会通过规则检查 Java Source，并将信息显示在 PMD 自己的视图中，如图 7-27 所示。

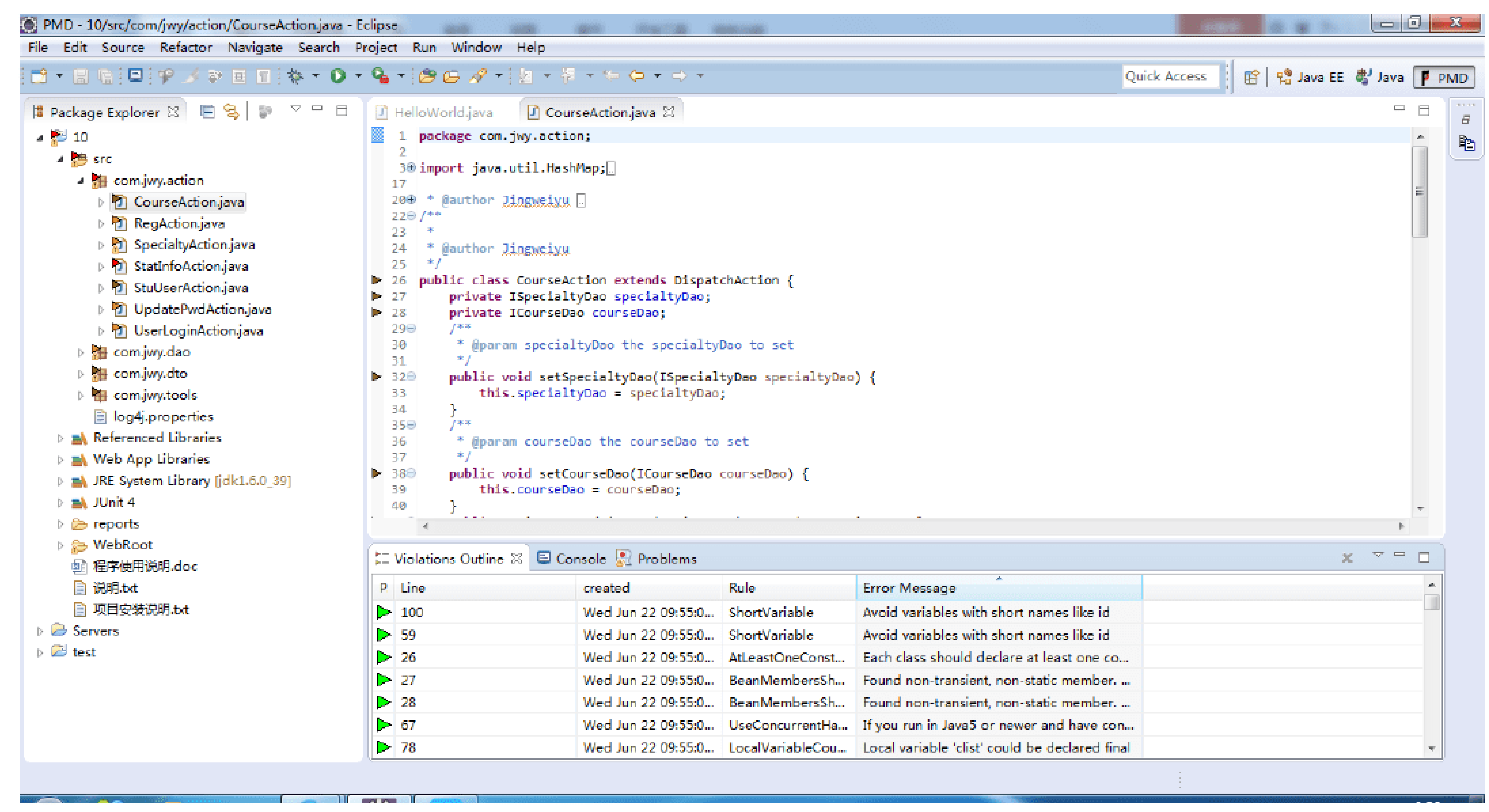


图 7-27　Java 源代码检查

PMD 会检查出：代码中出现 System.out.print 等警告描述。

(4) 检查代码重复

PMD 的 CPD 能够帮助发现代码的重复，如图 7-28 所示。

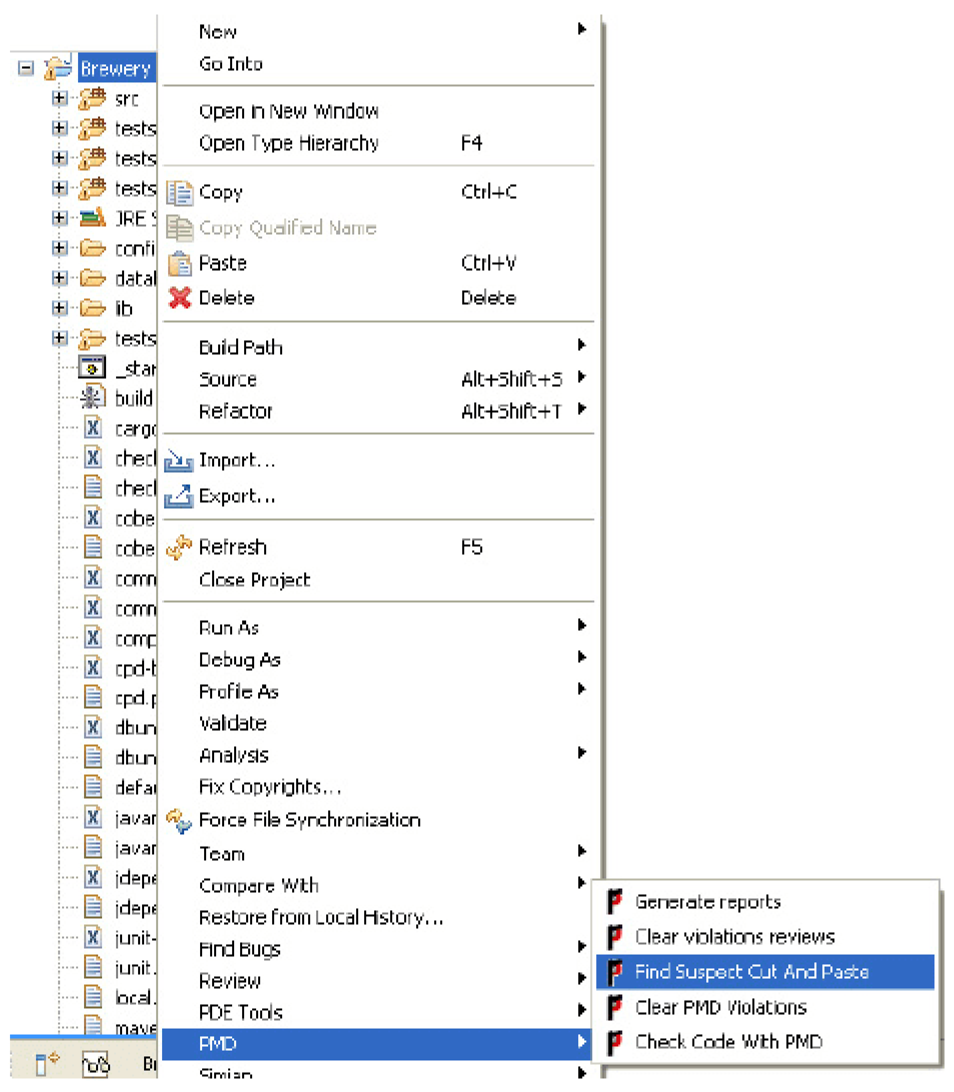


图 7-28　代码重复检查

一旦运行 CPD，在 Eclipse 根目录下就会创建出一个 report 文件夹，其中包含一个名为 cpd.txt 的文件，文件中列示了所有重复的代码，如图 7-29 所示。

d:\workspace\orderdeal\reports\cpd-report.txt - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R) 插件(P) 窗口(W) ?

cpd-report.txt | hosts | test.py | LocalSettings.php | FCKeditorSajax.body.php | ProfilerStub.php

```
Found a 237 line (735 tokens) duplication in the following files:
Starting at line 261 of D:\workspace\orderdeal\src\com\hollycrm\vo\CustOrderChange.j
Starting at line 293 of D:\workspace\orderdeal\src\com\hollycrm\vo\CustOrder.java

    public void setProcInsId(String procInsId) {
        this.procInsId = procInsId;
    }

    public Date getReqFinish() {
        return this.reqFinish;
    }

    public void setReqFinish(Date reqFinish) {
        this.reqFinish = reqFinish;
    }

    public Date getCompletedTime() {
        return this.completedTime;
    }
```

图 7-29　显示重复代码

4. 应用举例

可以用 PMD 对下面的源码进行分析，该源码中存在一些缺陷，如图 7-30 所示。

```
package pmd.test;
import java.io.*;
public class Test {
    public boolean copy(InputStream is, OutputStream os) throws IOException {
        int count = 0;
        byte[] buffer = new byte[1024];
        while ((count = is.read(buffer)) >= 0) {
        os.write(buffer, 0, count);
        }
        return true;
```

```
            }
        public void copy(String[] a, String[] b, String ending) {
            int index;
            String temp = null;
            int length = temp.length();
            for (index = 0; index <a.length; index++) {
                if (true) {
                    if (temp == ending) {
                        break;
                    }
                    b[index] = temp;
                }
            }
        }
        public void readFile(File file) {
            InputStream is = null;
            OutputStream os = null;
            try {
                is = new BufferedInputStream(new FileInputStream(file));
                os = new ByteArrayOutputStream();
                copy(is, os);
                is.close();
                os.close();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
        }
    }
```

```
public class Test {

    public boolean copy(InputStream is, OutputStream os) throws IOException {
        int count = 0;
        byte[] buffer = new byte[1024];
        while ((count = is.read(buffer)) >= 0) {
            os.write(buffer, 0, count);
        }
        return true;
    }

    public void copy(String[] a, String[] b, String ending) {
        int index;
        String temp = null;
        int length = temp.length();// 未使用变量和空指针错误
        for (index = 0; index <a.length; index++) {
            if (true) { // 多余的if语句
                if (temp == ending) { // 对象比较应使用equals
                    break;
                }
                b[index] = temp; // 缺少数组下标越界检查
            }
        }
    }
    public void readFile(File file) {
        InputStream is = null;
        OutputStream os = null;
        try {
            is = new BufferedInputStream(new FileInputStream(file));
            os = new ByteArrayOutputStream();
            copy(is, os); // 未使用方法返回值
            is.close();
            os.close();
        } catch (IOException e) {
            e.printStackTrace(); // 可能造成I/O流未关闭
        } finally {
            // 空的try/catch/finally块
        }
    }
}
```

图 7-30　源代码中存在的问题

图 7-30 显示源程序中有 7 个问题：

问题 1：第 28 行，应该提示空指标错误。

问题 2：第 28 行，变量 length 未使用，应该提示未使用变量。

问题 3：第 30 行，由于 if 判断条件为 true，因此这里应该提示多余的 if 语句。

问题 4：第 31 行，应该提示对象比较应使用 equals。

问题 5：第 34 行，应该提示缺少数组下标越界检查。

问题 6：第 48 行，应该避免直接使用 printStackTrace 函数，可能造成 I/O 流未关闭。

问题 7：第 50 行，应该提示空的 finally 代码块。

将上述源代码保存为 Test.java 文件，通过 PMD|Check Code 对其源码进行检测，检查结果如图 7-31 所示。

Violations Outline　Console　Problems

P	Line	cr...	Rule	Error Message
	48	Fri...	Us...	Prefer String...
	98	Fri...	Us...	Prefer String...
	101	Fri...	Sy...	System.out.p...
	50	Fri...	Av...	The String lit...
	47	Fri...	Av...	The String lit...
	53	Fri...	Av...	The String lit...
	34	Fri...	Co...	publicMetho...
	86	Fri...	Co...	publicMetho...
	24	Fri...	Co...	publicMetho...
	44	Fri...	Co...	publicMetho...
	74	Fri...	Co...	publicMetho...
	65	Fri...	Co...	publicMetho...
	110	Fri...	Co...	publicMetho...

图 7-31　PMD 检查出的问题

违反 PMD 检查规则的统计列表如图 7-32 所示。

Violations Overview

Element	# Violations	# Violations/...	# Violations/...	Project
CommentRequired	5	384.6	1.67	10
AtLeastOneConstructor	1	76.9	0.33	10
NullAssignment	2	153.8	0.67	10
com.jwy.dto	182	866.7	2.46	10
MethodArgumentCouldBeFinal	60	285.7	0.81	10
CommentRequired	101	481.0	1.36	10
MissingSerialVersionUID	5	23.8	0.07	10
ShortVariable	11	52.4	0.15	10
UncommentedEmptyConstructor	5	23.8	0.07	10
com.jwy.dao	255	1243.9	5.80	10
LawOfDemeter	65	317.1	1.48	10
UseStringBufferForStringAppends	6	29.3	0.14	10
MethodArgumentCouldBeFinal	24	117.1	0.55	10
AvoidInstantiatingObjectsInLoops	1	4.9	0.02	10
CommentRequired	44	214.6	1.00	10
AtLeastOneConstructor	4	19.5	0.09	10
ConfusingTernary	1	4.9	0.02	10
UnnecessaryLocalBeforeReturn	9	43.9	0.20	10
CommentSize	1	4.9	0.02	10
SystemPrintln	5	24.4	0.11	10
AvoidDuplicateLiterals	3	14.6	0.07	10
UnusedModifier	22	107.3	0.50	10
LocalVariableCouldBeFinal	35	170.7	0.80	10
OnlyOneReturn	3	14.6	0.07	10
SimplifyBooleanReturns	1	4.9	0.02	10
ShortVariable	31	151.2	0.70	10
com.jwy.action	292	713.9	8.11	10

图 7-32　PMD 规则的违反情况

从执行结果来看，问题 1、2、3、4、6、7 均被有效提示，只有问题 5“缺少数组下标越界检查”未被检查出来，这是由于 PMD 只是通过静态分析来获知代码错误。也就是说，在不运行 Java 程序的情况下报告错误。而在 Java 语言中，数组下标越界属于运行时错误，故无法通过 PMD 报告错误。

7.5.3　代码质量分析工具 SourceMonitor

SourceMonitor 是一款免费的软件，运行在 Windows 平台下。它可对用多种语言写成的代码进行度量，包括 C、C++、C#、Java、VB、Delphi 和 HTML，并且针对不同的语言，输出不同的代码度量值。

像其他代码度量工具一样，SourceMonitor 只关注代码，并为编码人员提供及时的反馈。它不是一款项目管理工具，不关注项目实施中从功能分析到设计编码，再到测试的整个过程。

1. C 语言度量值(C Metrics)

下面以 C 语言度量值为例，看看 SourceMonitor 度量值有哪些。

1) 总行数(Lines)：包括空行在内的代码行数。

2) 语句数目(Statements)：在 C 语言中，语句是以分号结尾的。分支语句 if，循环语句 for、while，跳转语句 goto 都被计算在内，预处理语句#include、#define 和#undef 也被计算在内，对其他的预处理语句则不作计算，#else 和#endif、#elif 和#endif 之间的语句将被忽略。

3) 分支语句比例(Percent Branch Statements)：该值表示分支语句占语句数目的比例，这里的分支语句指的是使程序不顺序执行的语句，包括 if、else、for、while 和 switch。

4) 注释比例(Percent Lines with Comments)：该值指示注释行(包括/*……*/和//……形式的注释)占总行数的比例。

5) 函数数目(Functions)：指示函数的数量。

6) 平均每个函数包含的语句数目(Average Statements per Function)：将总的函数语句数目除以函数数目即得到该值。

7) 函数圈复杂度(Function Complexity)：圈复杂度指示一个函数可执行路径的数目，以下语句为圈复杂度的值贡献 1：if/else/for/while 语句，三元运算符语句，if/for/while 判断条件中的&&或||，switch 语句，后接 break/goto/return/throw/continue 语句的 case 语句，catch/except 语句。

8) 函数深度(Block Depth)：函数深度指示函数中分支嵌套的层数。

对于其他语言，SourceMonitor 输出不同的度量值。例如，在 C++度量值中包括类的数目(Classes)，在 HTML 中包括各个标签的数目(HTML Tags)、超链接数目(Hyperlinks)等。

2. 度量值的呈现样式

SourceMonitor 从几个不同的视图层次展示度量值，包括项目视图、检查点视图和函数视图。

1) 项目视图(project view)

在 SourceMonitor 下建立项目必须在一个文件夹下进行，该文件夹下的源码文件可以被分成一个或几个检查点，项目视图下列出了各个检查点的度量值信息，如图 7-33 所示。

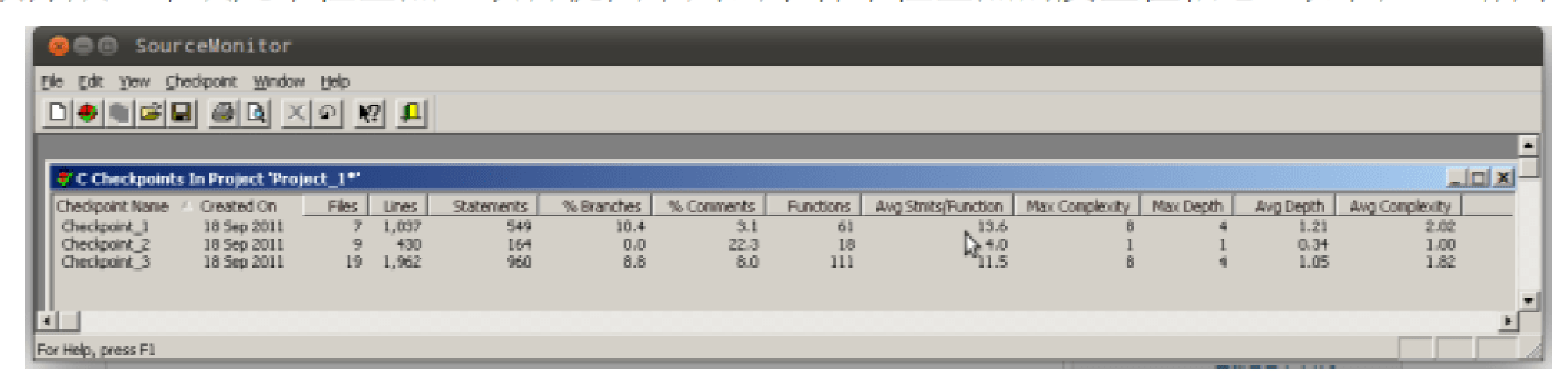


图 7-33　项目视图信息

2) 检查点视图(checkpoint view)

检查点视图列出了某个检查点中包含的各个源代码文件的度量值信息，如图 7-34 所示。

图 7-34　检查点视图信息

3) 函数视图(method view)，见图 7-35

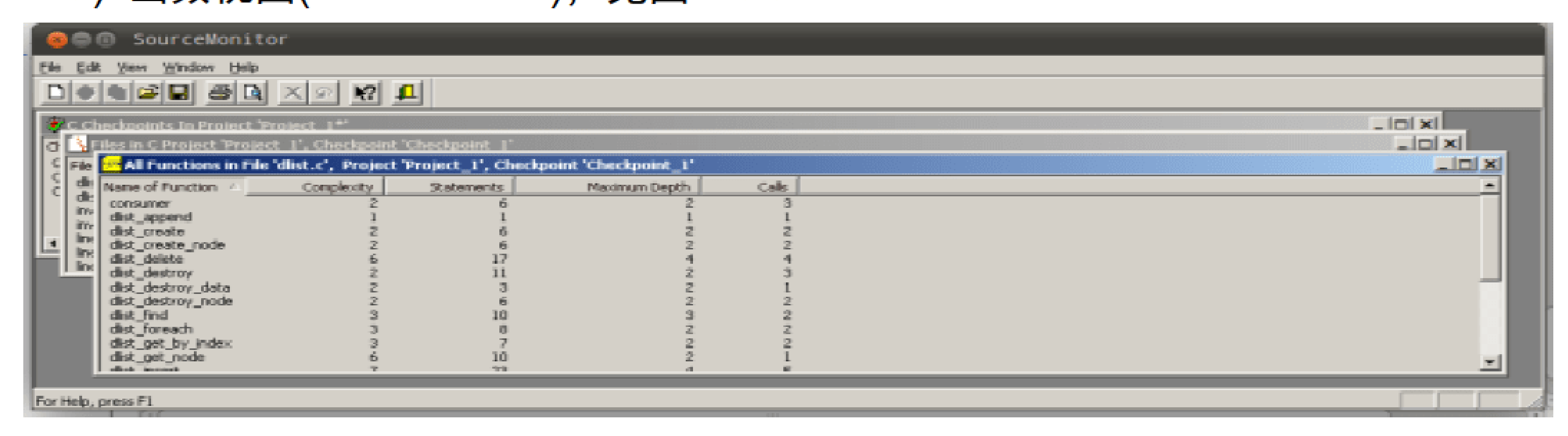


图 7-35　函数视图信息

3. 应用举例

1) 安装 SourceMonitor

SourceMonitor 是一个源代码衡量工具，下载链接为 http://www.campwoodsw.com/，可免费下载。下载软件安装包之后，进行默认安装。

2) 新建 SourceMonitor 项目

(1) 在 SourceMonitor 中新建工程，选择 Java 或 C++类型，单击“下一步”按钮。

(2) 指定源代码目录，如果有些目录不希望统计，可选择第 2 项排除个别目录。

(3) 指定工程名称、统计工程文件的保存位置，最后得出统计分析结果。

具体过程如下：

① 打开 SourceMonitor 软件，如图 7-36 所示。

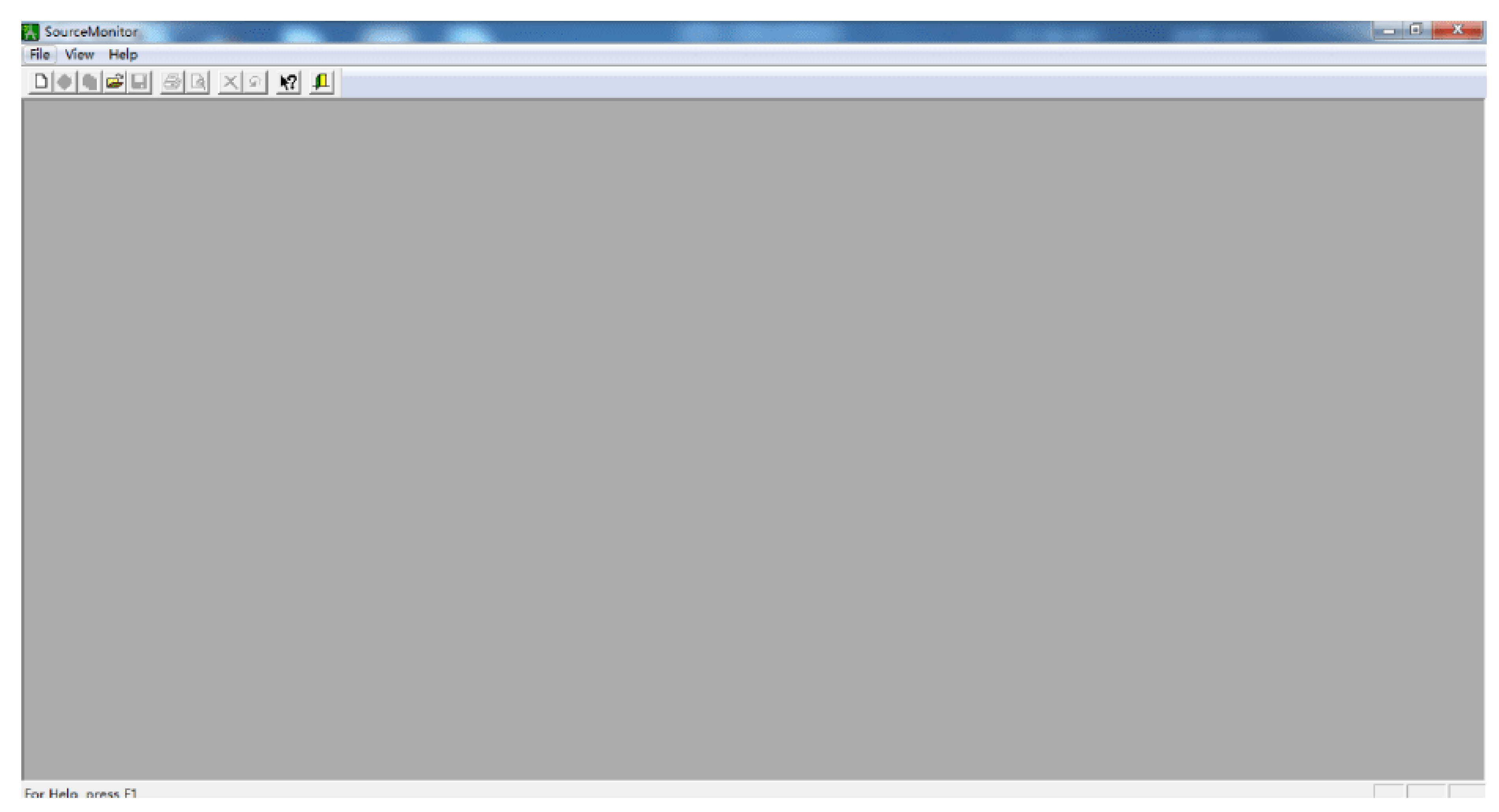


图 7-36　SourceMonitor 界面

② 选择 File | new，选择项目代码类型，因为我们要用 Java 代码来举例，所以在这里选择 Java 语言类型 如图 7-37 所示。

③ 确定项目名称和项目所在路径，如图 7-38 所示。

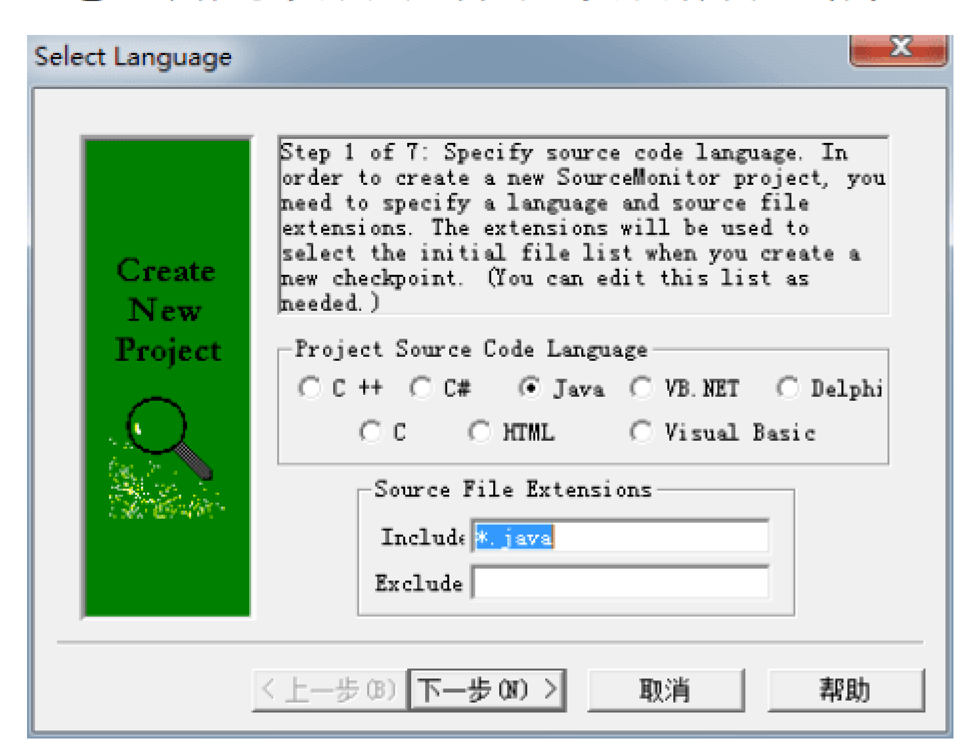


图 7-37　SourceMonitor 语言选择界面

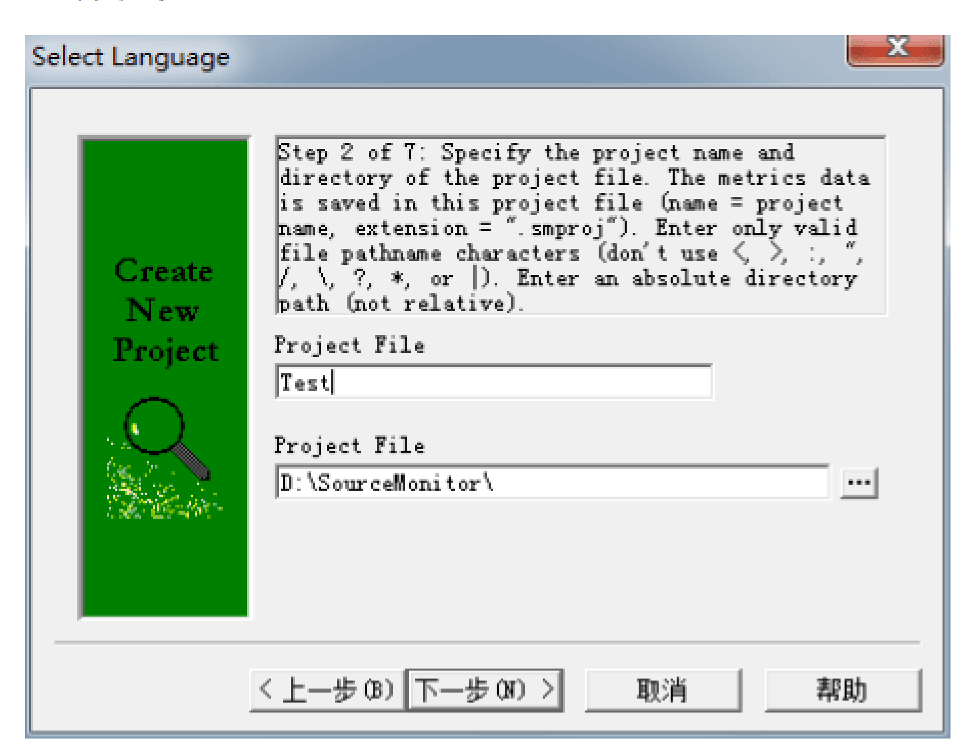


图 7-38　SourceMonitor 文件路径

④ 选择要进行代码分析的项目目录，主要是为了找到项目文件配置的 XML，如图 7-39 所示。

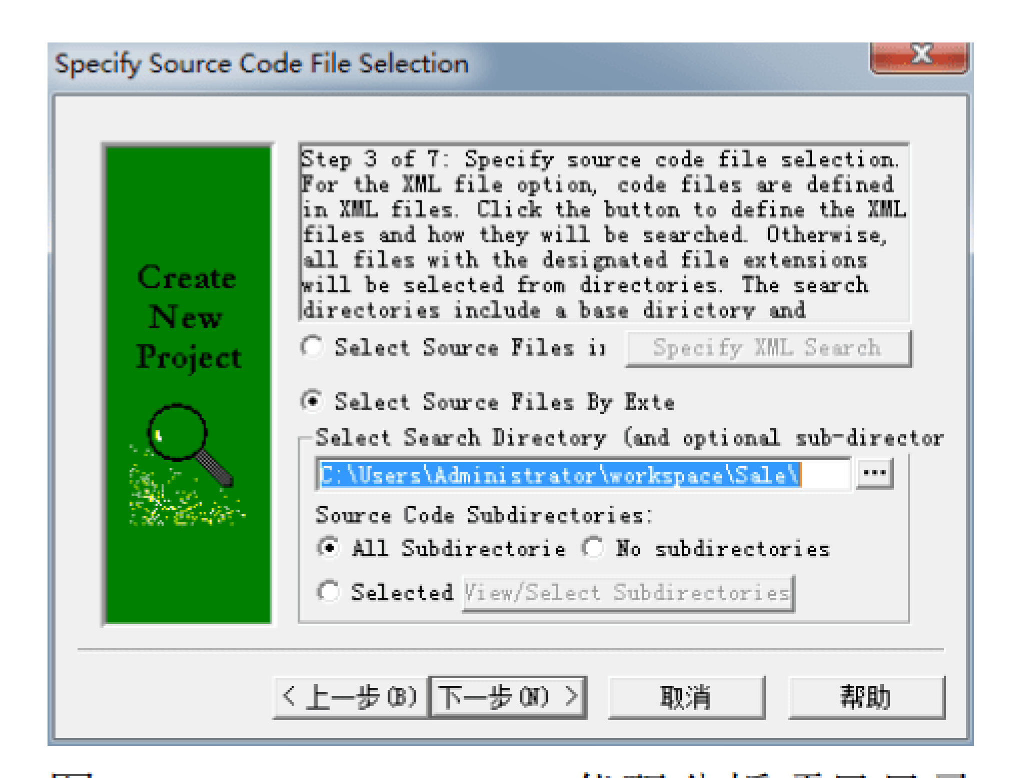


图 7-39　SourceMonitor 代码分析项目目录

然后一路单击“下一步”，每项均为默认选项，选择要检查的文件。

点击 OK，开始分析代码，并生成分析报告，如图 7-40 所示。

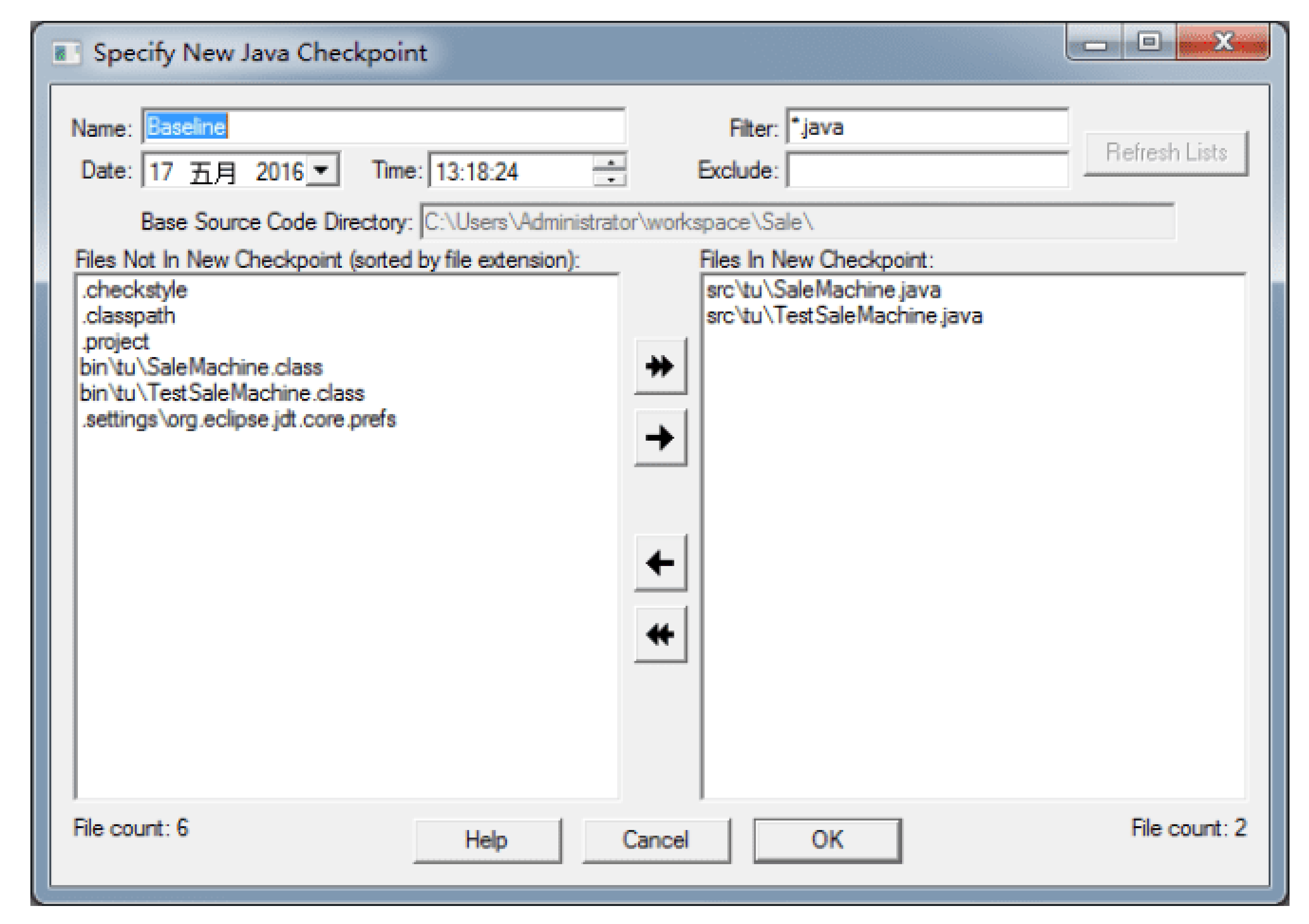


图 7-40　选择 SourceMonitor 检查文件

在 SourceMonitor 下建立项目必须在一个文件夹下进行，该文件夹下的源码文件可以被分成一个或几个检查点，项目视图中列出了各个检查点的度量值信息。

双击检查点信息，可以查看具体的的检查报告，通过这个报告，可以看到各方面的指标。检查点视图中列出了某个检查点中包含的各个源代码文件的度量值信息。例如：

- Files：代码文件数。
- Lines：代码行数，包含注释和空行。
- %Comment：注释量。
- Statements：纯代码行数。
- %Branches：分支数。
- Calls：函数调用数。
- Methods/Class：平均每个类中的方法。
- Classes：有几个类。
- Max Complexity：最大复杂度。
- Max depth：最大深度。

检查点视图如图 7-41 所示。

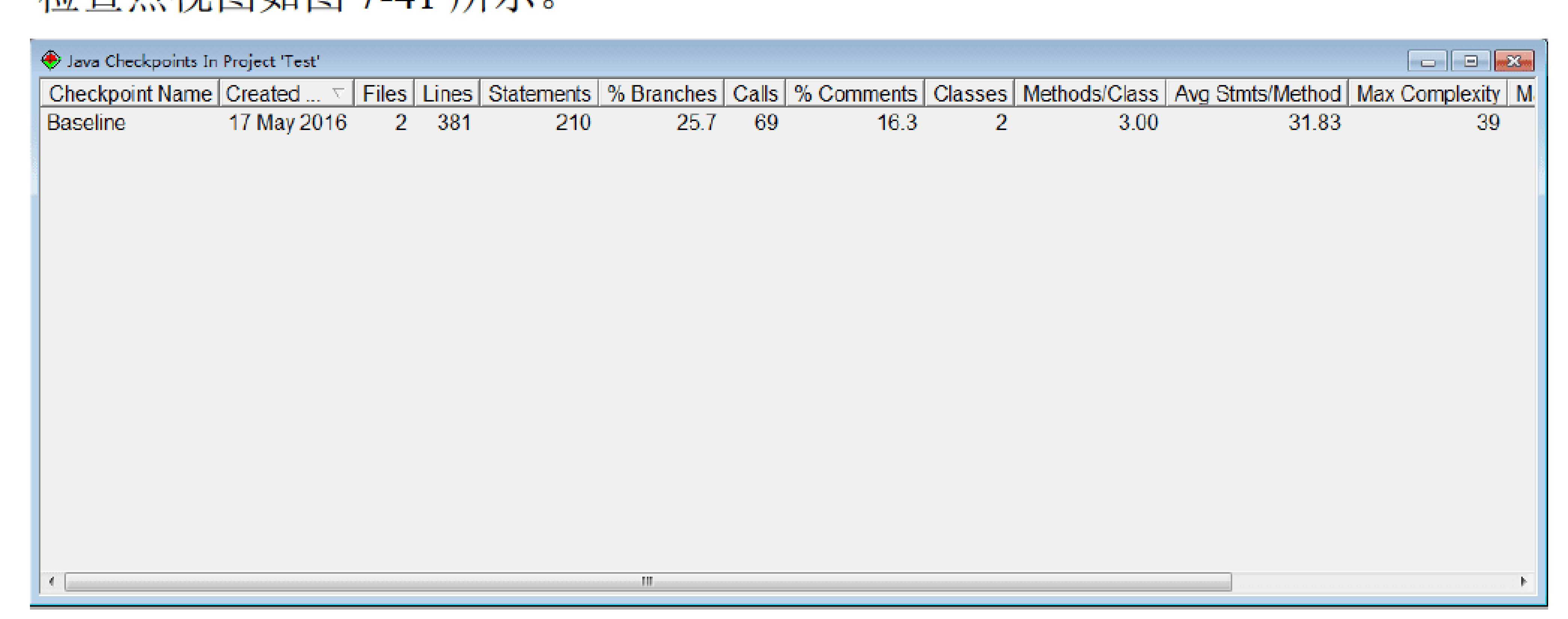


图 7-41　SourceMonitor 检查点视图

⑤ 双击某个检查点可以展开检查点下的各个函数。

⑥ 单击右键，可以出现查看具体条目的分析报告，以及可对源代码进行查看或修改等。

⑦ 函数视图中展示了在某个检查点下，某个源文件中所有函数的度量信息。函数视图如图 7-42 所示。

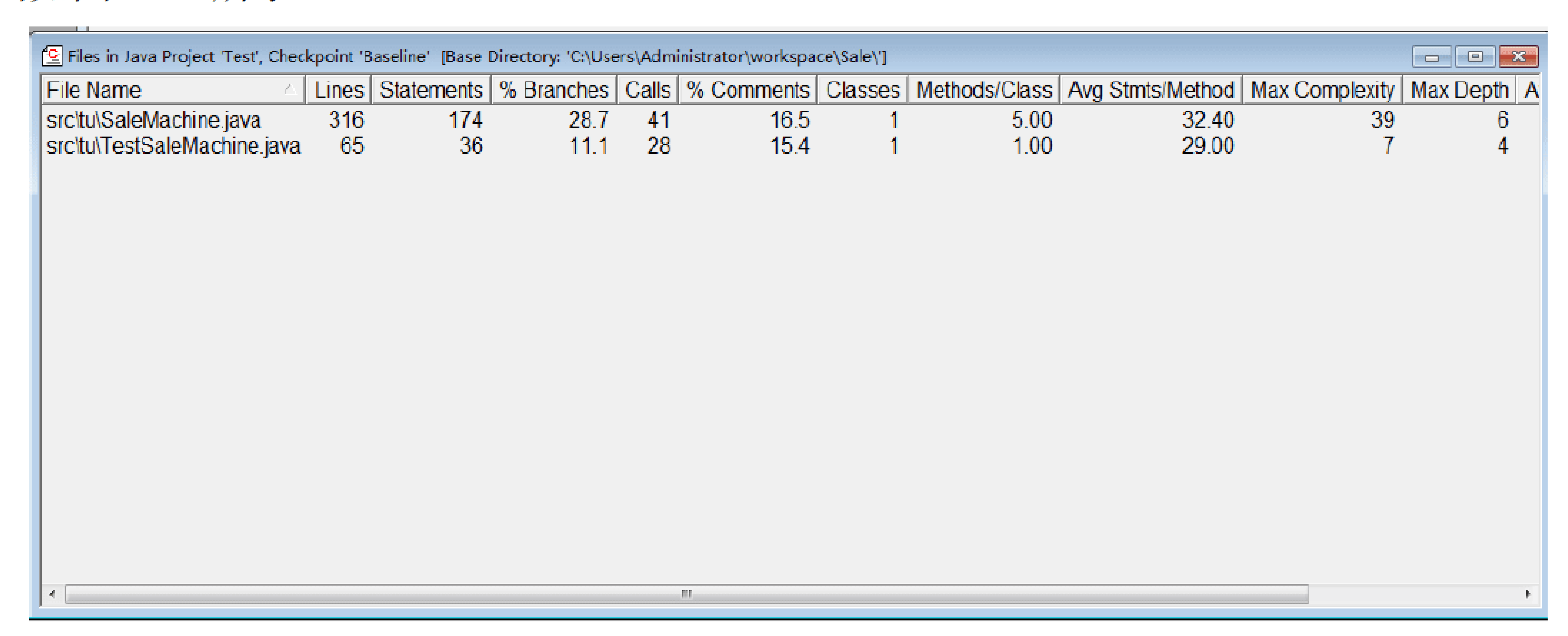


图 7-42　SourceMonitor 函数视图信息

如图 7-43 所示，在左下部分我们可以看到代码最主要关心的几个方面，能够一目了然看出哪些方面还需要改进，红色线表示当前的情况，在绿色范围(如平均复杂度在 2.0~4.0 之间)内表示良好。我们可以看到此段代码有两部分还不够良好。

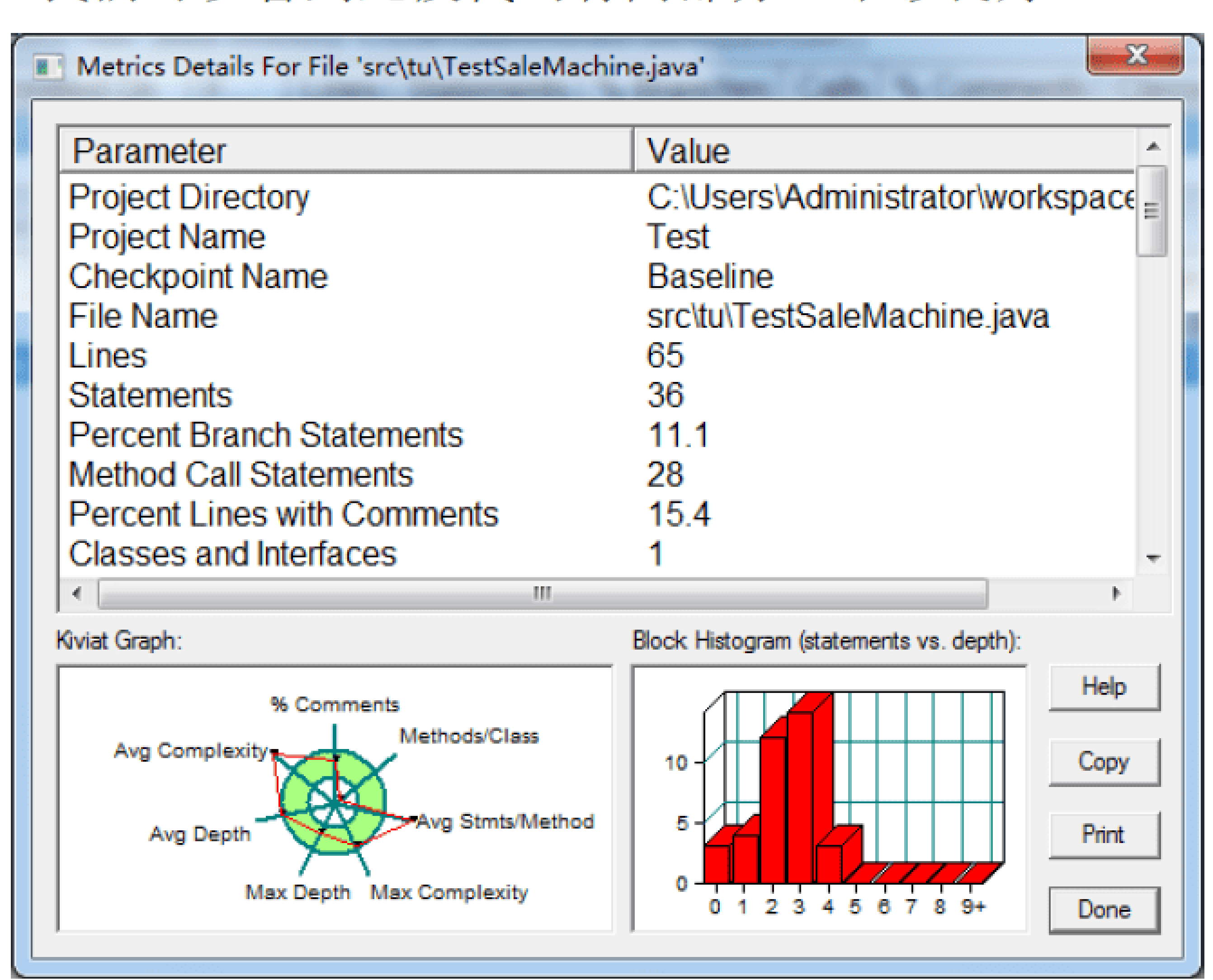


图 7-43　SourceMonitor 代码视图信息(一)

如图 7-44 所示，右下部分是代码行数和代码深度(嵌套的层数)的比例，根据分析报告，可以直接找到最复杂的文件和函数，这应该是首选的准备重构的文件。

根据以上情况可知：

(1) 代码度量工具 SourceMonitor 可以从几个不同的视图层次，为我们展示以上列举的度量值，包括项目视图、检查点视图和函数视图，如图 7-45 所示。

(2) 根据检验报告，可以知道项目中哪些类或函数需要重构，相比人工进行代码阅读，并确认重构代码部分，简单了很多，也可以对代码质量有最初的量化概念。

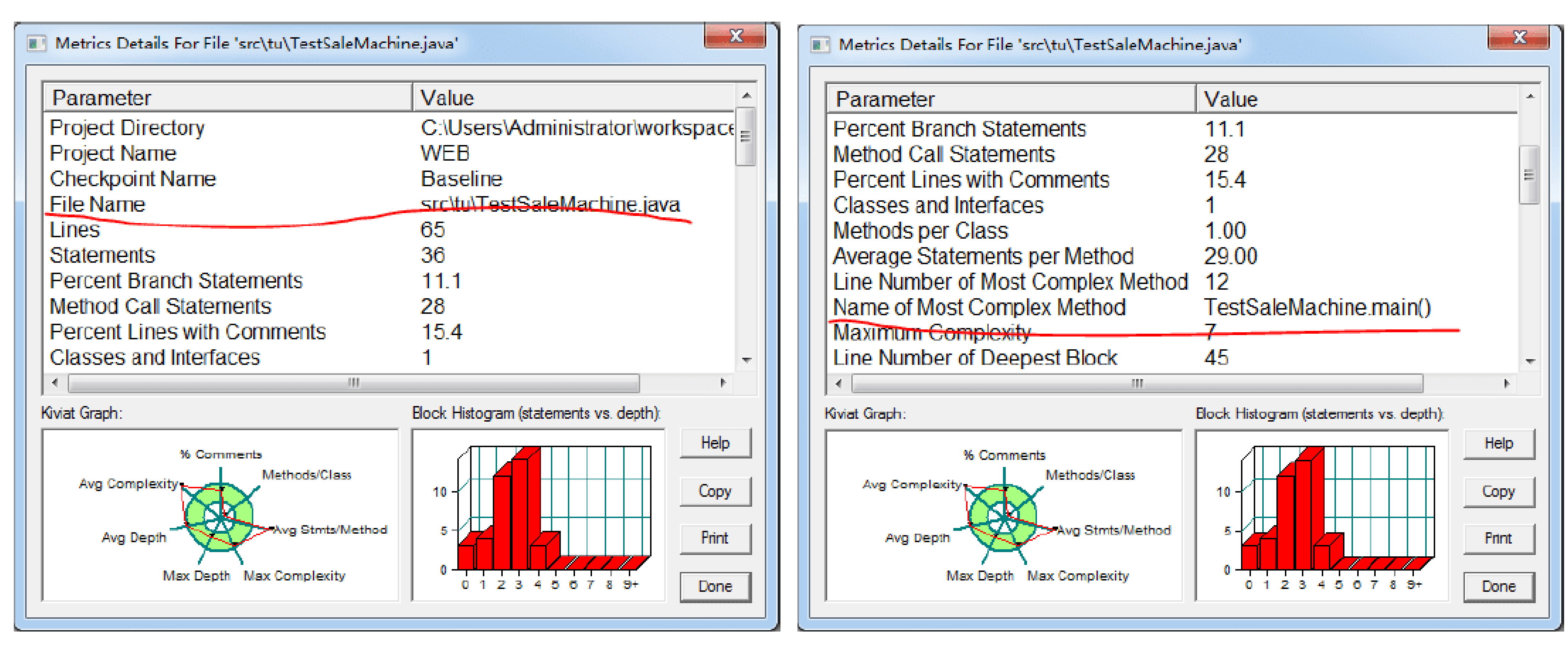


图 7-44　SourceMonitor 代码视图信息(二)

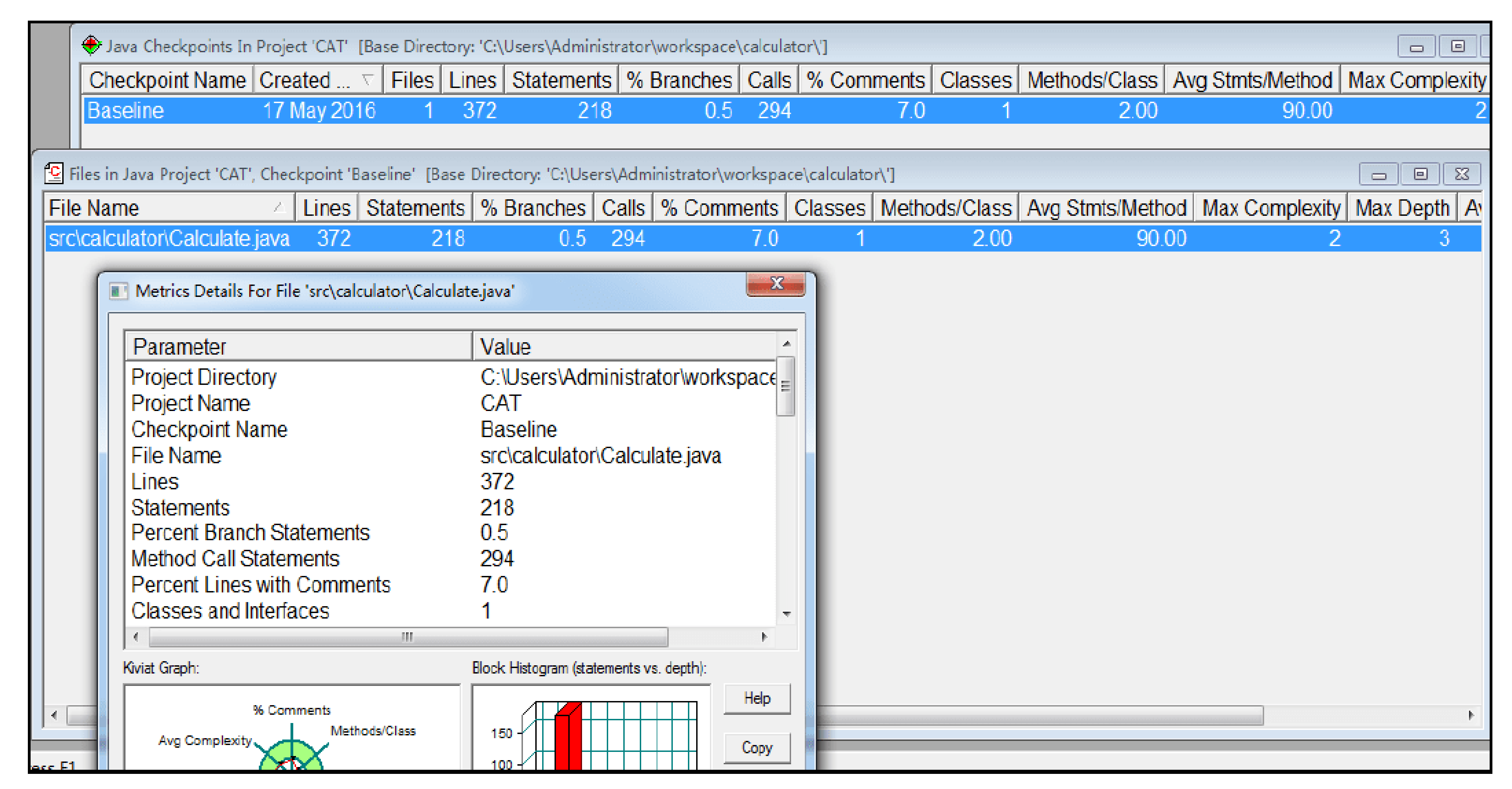


图 7-45　SourceMonitor 度量值视图信息

高校学生选课管理系统的代码质量分析结果如图 7-46~7-49 所示。

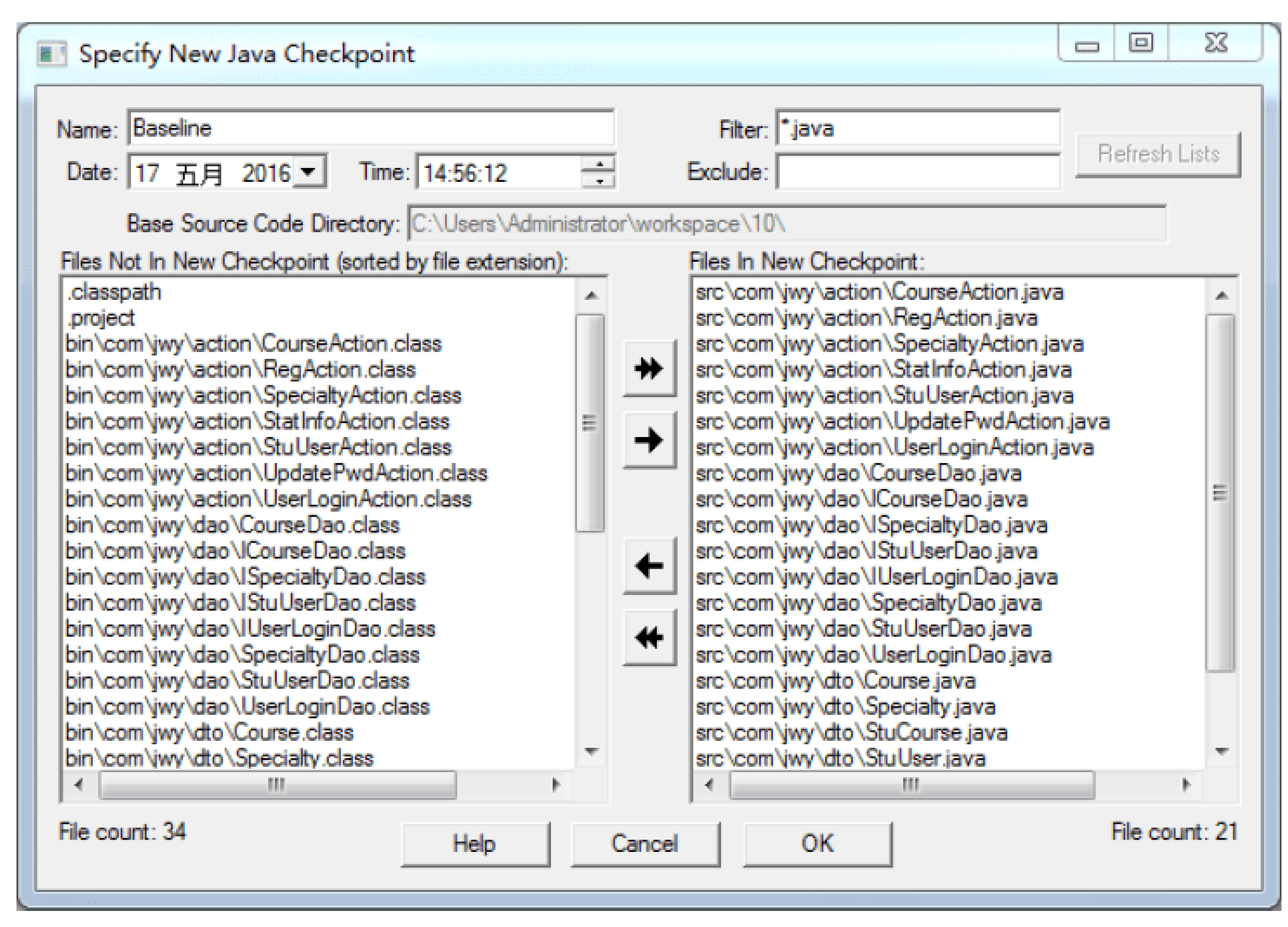
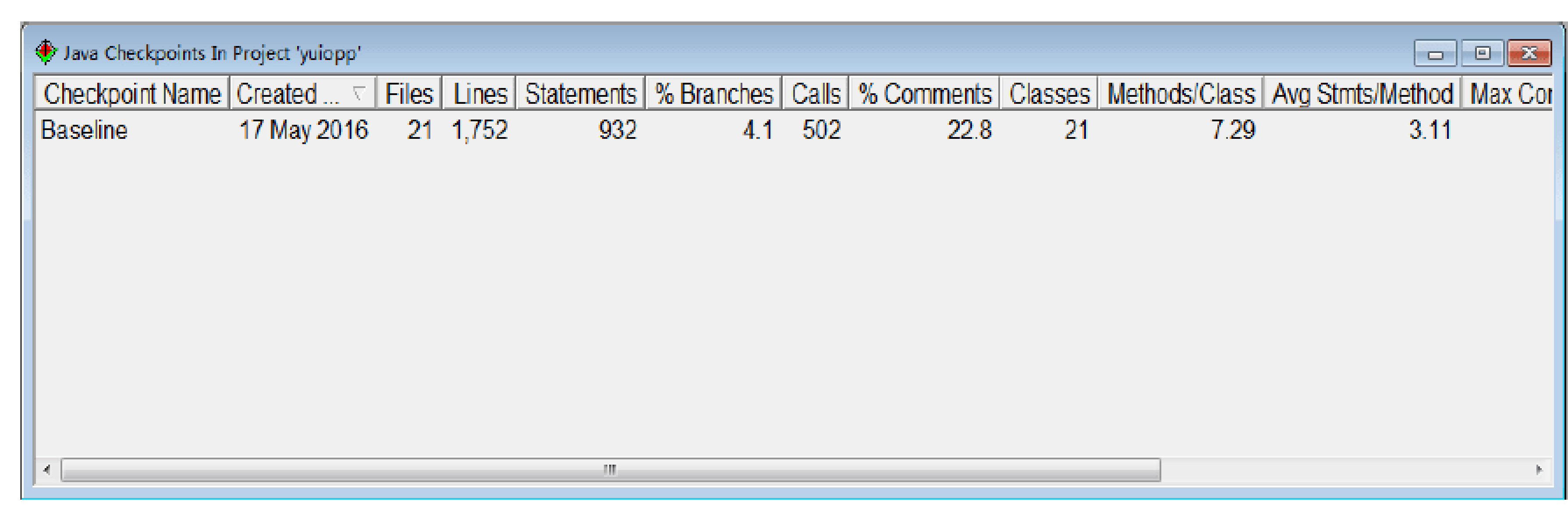


图 7-46　选择高校学生选课管理系统检查文件



Java Checkpoints In Project 'yuiopp'

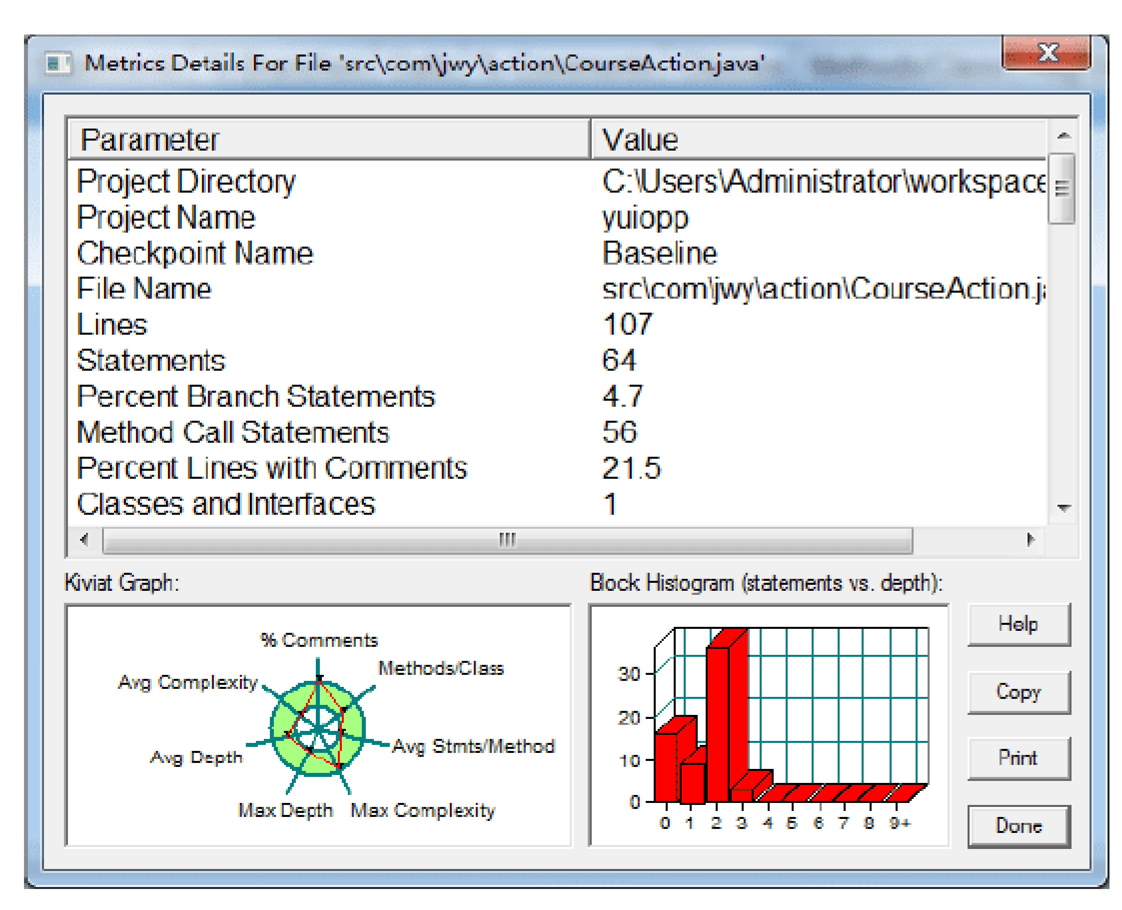
Checkpoint Name	Created ...	Files	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Co
Baseline	17 May 2016	21	1,752	932	4.1	502	22.8	21	7.29	3.11	

图 7-47　高校学生选课管理系统检查点视图

Files in Java Project 'yuiopp', Checkpoint 'Baseline' [Base Directory: 'C:\Users\Administrator\workspace\10\']

File Name	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth
src\com\jwy\action\CourseAction.java	107	64	4.7	56	21.5	1	7.00	5.57	7	3	1.41
src\com\jwy\action\RegAction.java	59	36	5.6	22	30.5	1	3.00	6.00	3	3	1.39
src\com\jwy\action\SpecialtyAction.java	65	36	0.0	21	20.0	1	4.00	4.75	1	2	1.19
src\com\jwy\action\StatInfoAction.java	295	183	2.7	186	27.1	1	6.00	19.33	7	3	1.59
src\com\jwy\action\StuUserAction.java	177	75	1.3	60	42.9	1	10.00	4.60	2	3	1.41
src\com\jwy\action\UpdatePwdAction.java	52	32	12.5	19	15.4	1	2.00	10.00	6	3	1.63
src\com\jwy\action\UserLoginAction.java	82	50	12.0	27	12.2	1	4.00	6.75	7	5	2.08
src\com\jwy\dao\CourseDao.java	117	54	11.1	33	22.2	1	5.00	2.80	1	3	1.56
src\com\jwy\dao\ICourseDao.java	20	12	0.0	0	55.0	1	5.00	0.00	0	1	0.58
src\com\jwy\dao\ISpecialtyDao.java	17	9	0.0	0	52.9	1	5.00	0.00	0	1	0.56
src\com\jwy\dao\IStuUserDao.java	17	9	0.0	0	52.9	1	5.00	0.00	0	1	0.56
src\com\jwy\dao\IUserLoginDao.java	16	8	0.0	0	56.3	1	5.00	0.00	0	1	0.63
src\com\jwy\dao\SpecialtyDao.java	70	30	0.0	14	27.1	1	5.00	2.40	1	2	1.13
src\com\jwy\dao\StuUserDao.java	98	52	7.7	29	20.4	1	5.00	6.20	3	4	1.67
src\com\jwy\dao\UserLoginDao.java	95	52	11.5	31	21.1	1	5.00	6.80	3	3	1.69
src\com\jwy\dto\Course.java	121	63	0.0	0	11.6	1	22.00	1.32	1	2	1.43
src\com\jwy\dto\Specialty.java	81	33	0.0	0	6.2	1	12.00	1.17	1	2	1.36
src\com\jwy\dto\StuCourse.java	47	21	0.0	0	14.9	1	8.00	1.00	1	2	1.29
src\com\jwy\dto\StuUser.java	109	58	0.0	0	11.9	1	20.00	1.35	1	2	1.43
src\com\jwy\dto\UserLogin.java	69	33	0.0	0	7.2	1	12.00	1.17	1	2	1.36
src\com\jwy\tools\CharacterEncodingFilter.java	38	22	4.5	4	10.5	1	3.00	2.67	2	3	1.05

图 7-48　高校学生选课管理系统函数视图



Metrics Details For File 'src\com\jwy\action\CourseAction.java'

Parameter	Value
Project Directory	C:\Users\Administrator\workspace
Project Name	yuiopp
Checkpoint Name	Baseline
File Name	src\com\jwy\action\CourseAction.j
Lines	107
Statements	64
Percent Branch Statements	4.7
Method Call Statements	56
Percent Lines with Comments	21.5
Classes and Interfaces	1

图 7-49　高校学生选课管理系统质量分析视图

习题和思考题

1. 什么是静态测试？静态测试包括哪些内容？
2. 什么是同行评审？简述同行评审的内容和流程。
3. 什么是需求规格说明测试？我们如何对需求规格说明进行评审？
4. 什么是代码审查？代码审查包括哪些内容？
5. 代码检查包括哪些内容？我们如何进行代码检查？

6. 什么是编码规范？确立和遵守编码规范有何意义？

7. 代码分析工具是怎样工作的？代码自动分析都有哪些内容？简单介绍几款针对不同语言的编程规则检查工具(在网上查找)。

8. 什么是代码结构分析？代码结构分析有何意义？

9. 从网上下载能绘制程序控制流图和程序调用图的软件，并给出实际使用的例子和结果。

10. 什么是代码安全性检查？简述代码安全性检查的方法和内容。

11. 什么是软件复杂性？软件复杂性包括哪些内容？

12. 什么是 Halstead 复杂度？Halstead 复杂度度量的主要思想是什么？

13. McCabe 复杂度的中心思想是什么？我们如何进行 McCabe 复杂度的度量？

14. 简述软件复杂性度量的基本方法，可采取何种手段控制软件复杂性？

15. 面向对象软件复杂性度量的特性有哪些？我们一般用于度量的方法有哪些？

16. 简述软件质量定义，软件质量属性包括哪些内容？

17. 简述软件质量分层模型的概念，目前流行的质量分层模型有哪些？

18. GB/T 16260-2006(软件工程产品质量标准)质量模型中的质量特性/子特性有哪些？我们一般如何处理质量特性/子特性之间的冲突？

19. 分别应用 PMD 和 CheckStyle 对 Java 程序代码进行分析，并总结它们各自的特点。

第8章　软件动态测试

动态测试是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能，这种方法由三部分组成：构造测试实例、执行程序、分析程序的输出结果。

对于动态测试，可以从不同的角度进行分类。比如：从是否关心软件内部结构和具体实现的角度划分，软件测试可以分为白盒测试、黑盒测试和灰盒测试；从软件开发过程的角度划分，软件测试可以分为单元测试、集成测试、确认测试、系统测试、验收测试及回归测试；从测试执行时是否需要人工干预的角度划分，软件测试可以分为人工测试和自动化测试；从测试实施组织的角度划分，软件测试可分为开发方测试、用户测试(β 测试)、第三方测试。

8.1　白盒测试

白盒测试是一种典型的测试方法，是一种按照程序内部逻辑结构和编码结构设计测试数据并完成测试的测试方法，因此又称为结构测试或逻辑驱动测试。白盒测试基于应用代码的内部逻辑知识，测试覆盖全部代码、分支、路径和条件。它利用查看代码功能和实现方式得到的信息来确定哪些需要测试、哪些不需要测试以及如何展开测试。

白盒测试一般分为静态测试和动态测试，静态测试不实际运行软件，主要是对软件的编程格式、结构等方面进行评估，采用的是代码走查、代码审查、程序结构分析、控制流分析、数据流测试及信息流分析等；而动态测试需要在 Host 环境或 Target 环境中实际运行软件，并使用设计的测试用例去探测软件缺陷。所采用的测试方法是逻辑覆盖(包括语句覆盖、分支覆盖、条件覆盖、分支-条件覆盖以及路径覆盖)。需要注意的是，不要把白盒测试和调试弄混了。调试和白盒测试都包括处理软件缺陷和查看代码的过程，但是它们的目标不同，其中又有交叉，如图 8-1 所示。

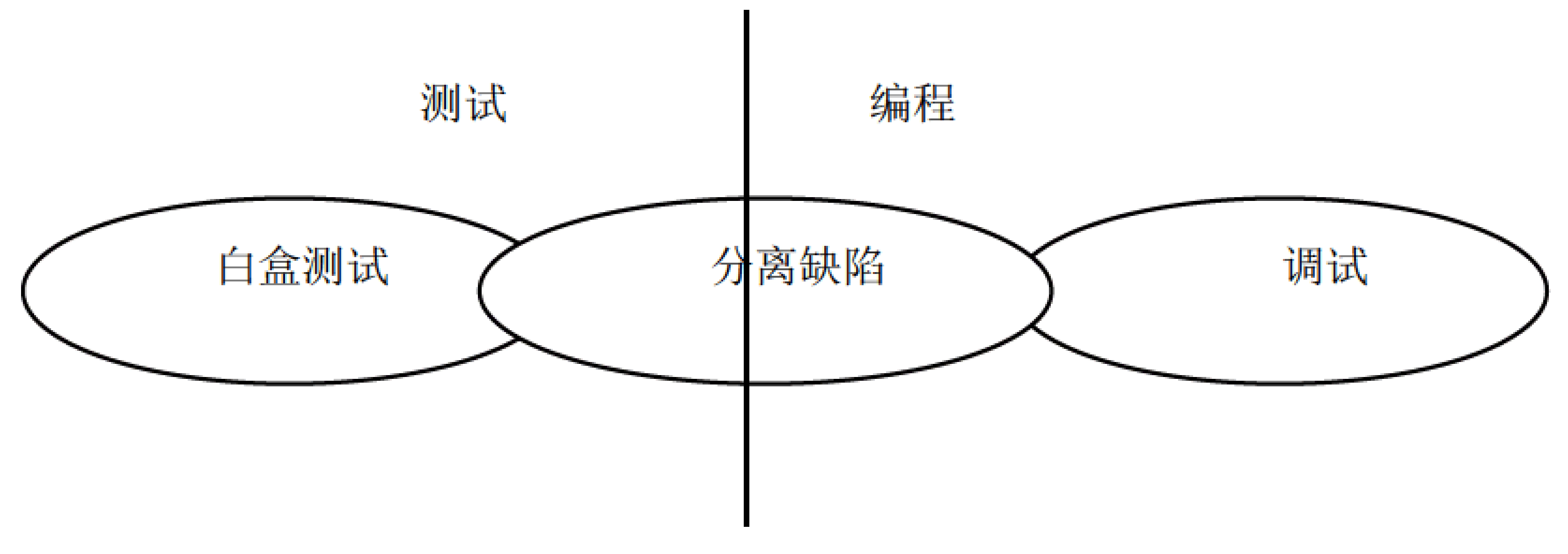


图 8-1　调试和白盒测试的目标不同，但有交叉

由图 8-1 可以看出白盒测试的目的是发现问题，而调试的目的是改正缺陷，但它们共

同的目的是分离缺陷。

白盒测试的特点主要有：①可以构成测试数据，使特定程序部分得到测试；②有一定的充分性度量手段；③可获得较多工具的支持；④通常只用于单元测试。

白盒测试的内容有：①对程序模块的所有独立执行路径至少测试一次；②对所有的逻辑判定，取真与取假的两种情况都至少测试一次；③在循环的边界和运行的边界限制内执行循环体；④测试内部数据结构的有效性。

我们用例 8-1 讲述白盒测试的方法。

例 8-1

```
func(int a, b, x)
{
      if ((a>1) && (b=0))
          x=x/a;
      if ((a=2) || (x>1))
          x=x+1;
}
```

该程序的流程图如图 8-2 所示。

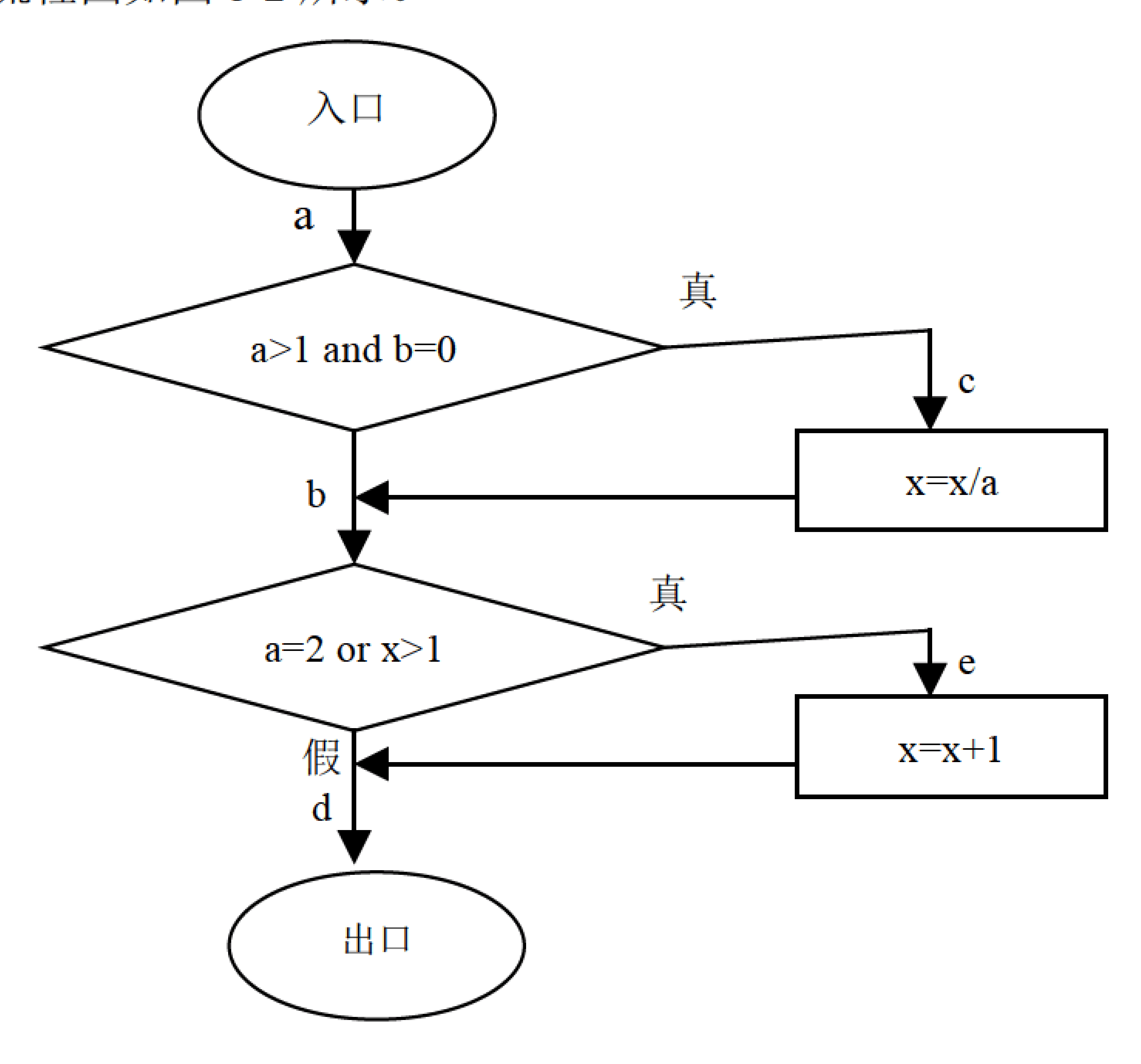


图 8-2　示例程序的流程图

8.1.1　逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础的测试方法，属于白盒测试。这一方法是一系列测试过程的总称，要求测试人员对程序的逻辑结构有清楚的了解。从覆盖源程序的各个方面考虑，大致可以分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。

1. 语句覆盖

为了暴露程序中的错误，语句覆盖是最起码的测试要求，要求设计足够多的测试用例，使得每一条语句至少被执行一次。对于例 8-1 来说，只要选取 a=2、b=0、x=3，就可以达到每一条语句至少执行一次的要求。

语句覆盖对程序的逻辑覆盖很少，在上面的例子中，两个判定条件都只测试条件为真的情况，条件为假时如果处理有误，显然不能发现。此外，语句覆盖只关心判定表达式的值，而没有分别测试判定表达式中每个条件取不同值时的情况。在上面的例子中，为了满足语句覆盖测试的要求，只需要两个判定表达式(a>1) and (b=0)和(a=2) or (x>1)都取真值，因此使用上述一组测试数据就够了。但是如果把程序中的第一个判定表达式中的逻辑运算符 and 写成 or，或者把第二个判定表达式中的条件写成 x<1，使用上面的测试数据则不能查出这些错误。

因此，语句覆盖是很弱的逻辑覆盖标准，为了更充分地测试程序，可以采用下面讲述的其他逻辑覆盖方法。

语句覆盖的优点：①检查所有语句；②结构简单的代码的测试效果较好；③容易实现自动测试；④代码覆盖率高；⑤如果是程序块覆盖，则不用考虑程序块中的源代码。

语句覆盖不能检查出的错误有：

(1) 条件语句错误(如 A > 1 && B == 0 -> A > 0 && B == 0)。

(2) 逻辑运算(&&、||)错误(如 A > 1 && B == 0 -> A > 1 || B == 0，以及 U = A<1 || B>2 -> U = A<1)。

(3) 循环语句错误，如循环次数错误(for(i=0；i<10；i++) statement；　-> for(i=0；i<=10；i++)　statement;) 以及跳出循环条件错误(While(x>3)　statement；　-> (While(x>3&&x<7) statement;)。

2. 判定覆盖(分支覆盖)

判定覆盖又叫分支覆盖，要求设计足够多的测试用例，使得程序中的每一个分支至少通过一次，即每一条分支语句的真值和假值都至少执行一次。while 语句、switch 语句、异常处理、跳转语句和三目运算符(a?b:c)等同样可以使用分支覆盖来测试。对于例 8-1 的流程图分支来说，设计两个测试用例即可使它能通过 acd 和 abe 路径就达到分支覆盖。

例如：(1) a=3，b=0，x=1 (沿 acd 路径执行)

　　　(2) a=2，b=1，x=3 (沿 abe 路径执行)

除了双分支语句外，还有多分支语句，如 C 语言中的 case 语句，分支覆盖必须对每一个分支的每一种可能的结果都进行测试，但是上面的测试用例没有检查 abd 路径执行时，x 的值是否有变化。因此，判定覆盖虽然比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高。

判定覆盖要比语句覆盖的查错能力强一些：执行了分支覆盖，实际也就执行了语句覆盖。判定覆盖与语句覆盖存在同样的缺点：不能查出条件语句错误，不能查出逻辑运算错误，不能查出循环次数错误，不能查出循环条件错误。

3. 条件覆盖

条件覆盖的含义是，不仅每一条语句至少执行一次，而且使得判定中的每个条件获得各种可能结果。

在例 8-1 所示程序中，(a>1)&& (b=0)包含两个条件：a>1，b=0。(a=2)|| (x>1)包含两个条件：a=2，　x>1　。因此，流程图中共有四个条件：a>1，b=0，a =2，x>1。为了达到条件覆盖的要求，需要有足够的例子来满足在 a 点有 a>1、a<=1、b=0、b!=0 等各种结果，在 b 点有 a=2、a!=2、x>1、x<=1 等各种结果。因此可设计以下两个测试用例来满足这一标准：

例如：(1) a=2，b=0，x=4(沿 ace 路径执行)

(2) a=1，b=1，x=1(沿 abd 路径执行)

同样是两个测试用例，但是这两个测试用例比分支覆盖中的更有效。因为它使判定表达式中每个条件都取得两个不同的结果，判定覆盖只关心整个判定表达式的结果。例如，上面两组测试数据也同时满足了判定覆盖的要求。但是，也可能出现相反的情况：虽然每个条件都取得不同的结果，判定表达式却有可能始终都是一个值。例如，如果使用下面两组测试用例，则只满足条件覆盖的要求而不满足判定覆盖的要求：

例如：(1) a=2，b=0，x=1(满足 a>1、b=0、a=2 和 x<=1 的条件，执行路径 ace)

(2) a=1，b=1，x=2(满足 a<=1、b!=0、a!=2 和 x>1 的条件，执行路径 abe)

条件覆盖的利弊：①能够检查所有的条件错误；②不能实现对每个分支的检查；③测试用例数增加。

4. 判定/条件覆盖

既然条件覆盖不一定包括判定覆盖，判定覆盖也不一定包括条件覆盖，自然会提出一种能够同时满足这两种覆盖标准的逻辑覆盖，这就是判定/条件覆盖。判定/条件覆盖就是设计足够多的测试用例，使得判定中每个条件的所有可能取值至少能够执行一次，同时每个判断的所有可能的判定结果至少执行一次。换言之，即要求各个判定的所有可能的条件取值组合至少执行一次。

对于例 8-1 中的程序而言，下述两组测试用例可以满足判定/条件覆盖的要求：

例如：(1) a=2，b=0，x=4

(2) a=1，b=1，x=1

但是，这两组测试用例也就是为了满足条件覆盖标准最初选取的两组数据。因此，有时候判定/条件覆盖也并不比条件覆盖强。

分支-条件覆盖的利弊：①既考虑每一个条件，又考虑每一个分支，发现错误的能力强于分支覆盖和条件覆盖；②并不能全面覆盖所有路径；③用例数量的增加。

5. 条件组合覆盖

要求设计足够多的测试用例，使得每个判定中条件的各种组合至少出现一次。

对于例 8-1 来说，程序中有四个条件：a>1，b=0，a=2，x>1。因此，设计测试用例时，应满足下面八种条件组合：

①a>1，b=0；②a>1，b!=0；③a<=1，b=0；④a<=1，b!=0；⑤a=2，x>1；⑥a=2，x<=1；⑦a!=2，x>1；⑧a!=2，x<=1。

例如：(1) a=1，b=0，x=2 满足③⑦组合
(2) a=1，b=1，x=1 满足④⑧组合
(3) a=2，b=0，x=4 满足①⑤组合
(4) a=2，b=1，x=1 满足②⑥组合

显然，满足条件组合覆盖标准的测试用例，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此，条件组合覆盖是前面几种覆盖标准中最强的。但是，满足条件组合覆盖要求的测试用例并不一定能使程序中的每条路径都执行到，例如，上述 4 组测试用例都没有测试到路径 acbd。

以上根据测试数据对源程序语句检测的详尽程度，简单讨论了几种逻辑覆盖标准。在上面的分析过程中，常常谈到测试数据执行的程序路径，显而易见，测试数据可以检测的程序路径的多少，也在一定程度上反映了对程序测试的详尽程度，也就是下面要讲的路径覆盖。

6. 路径覆盖

要求设计足够多的测试用例，使得程序中所有的路径都至少执行一次。对于例 8-1 中的流程图来说，有 4 条路径 ace、abd、abe 和 acd，因此设计了下面四个测试用例：

例如：(1) a=2，b=0，x=3　覆盖 ace
(2) a=2，b=1，x=1　覆盖 abe
(3) a=1，b=0，x=1　覆盖 abd
(4) a=3，b=0，x=1　覆盖 acd

8.1.2　路径测试

路径测试就是根据程序的逻辑控制所产生的路径进行测试用例设计的方法。它是从一个程序的入口开始，执行所经历的各条语句的完整过程。从广义的角度讲，任何有关路径分析的测试都可以被称为路径测试。

完成路径测试的理想情况是做到路径覆盖，但对于复杂性高的程序要做到所有路径覆盖(测试所有可执行路径)是不可能的。

在不能做到所有路径覆盖的前提下，如果某一程序的每一条独立路径都被测试过，那么可以认为程序中的每条语句都已经检验过了，即达到了语句覆盖。这种测试方法就是通常所说的基本路径测试方法。

下面介绍几种常用的路径测试方法。

1. DD 路径测试

DD路径(Decision-to-Decision Path)主要着眼命令式程序语言的测试覆盖率问题。程序有向图中存在分支，覆盖率考虑的是对各个分支情况的测试覆盖程度，因此对有向图中线性串行的部分进行压缩，在压缩图(即DD-路径)的基础上进行测试用例设计，用测试覆盖指标考查测试效果。

例如，有向图如图 8-3 所示。

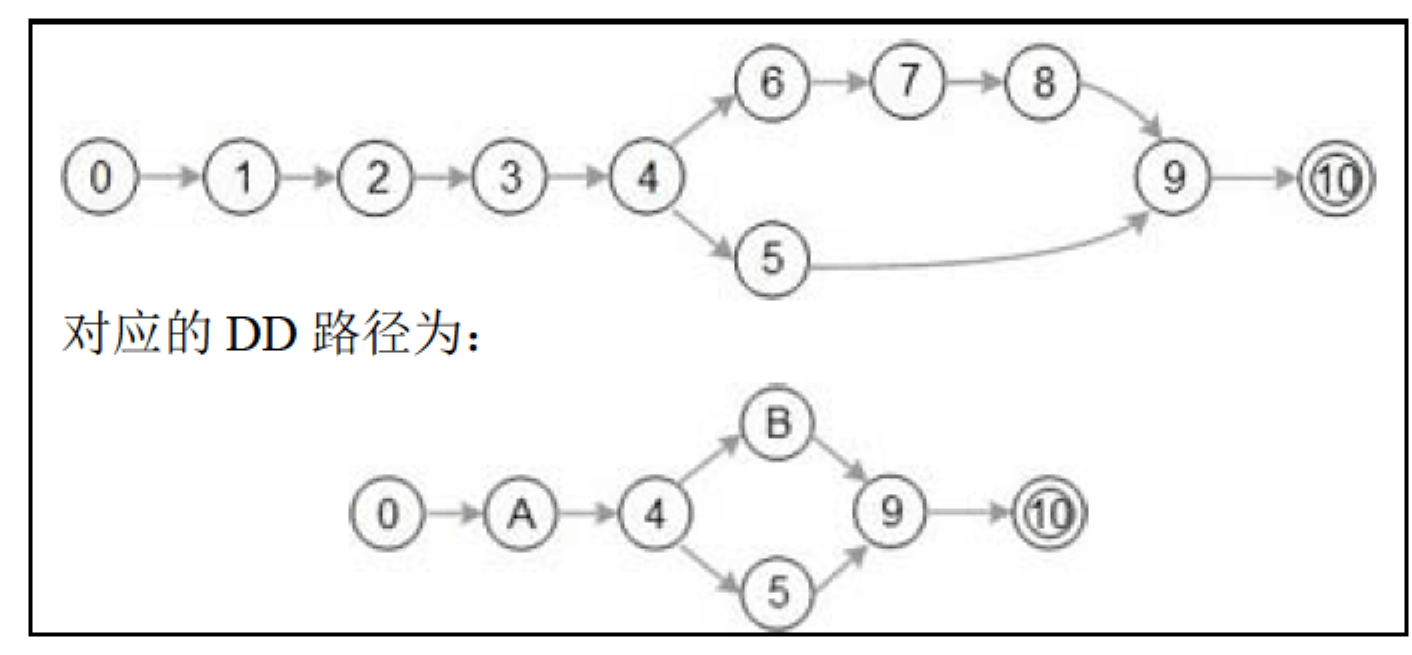


图 8-3　有向图到 DD-路径图的转换

即，将节点 1、2、3 合并成节点 A，将节点 6、7、8 合并成节点 B。合并原则为：将一系列邻接的顺序语句的节点合并。这样压缩的目的是将程序执行的分支情况清晰地提取出来，便于覆盖率的分析。

提出 DD 路径的目的：很多质量机构都把 DD 路径覆盖作为测试覆盖的最低可接受级别。E.F.Miller 发现，当通过一组测试用例满足 DD 路径覆盖要求时，可以发现全部缺陷中的大约 85%(Miller, 1991 年)。

如果每一条 DD 路径都被遍历，则我们知道每个判断分支都被执行，其实就是遍历 DD 路径图中的每条边。对于 if 类语句，这意味着真、假分支都要覆盖。对于 case 语句，则每条子句都要覆盖。

2. 基本路径测

例 8-1 是一个非常简单的程序段，只有 4 条路径。但是在实际问题中，一个不太复杂的程序，其路径都是一个非常庞大的数字。例如，图 8-4 所示的程序竟有 510 条路径。要想在测试中覆盖许许多多的路径是不现实的。为了解决这一难题，只得把覆盖的路径数据压缩到一定限度内。例如，程序中的循环体只执行一次。这里所介绍的基本路径测试就是这样一种测试方法。

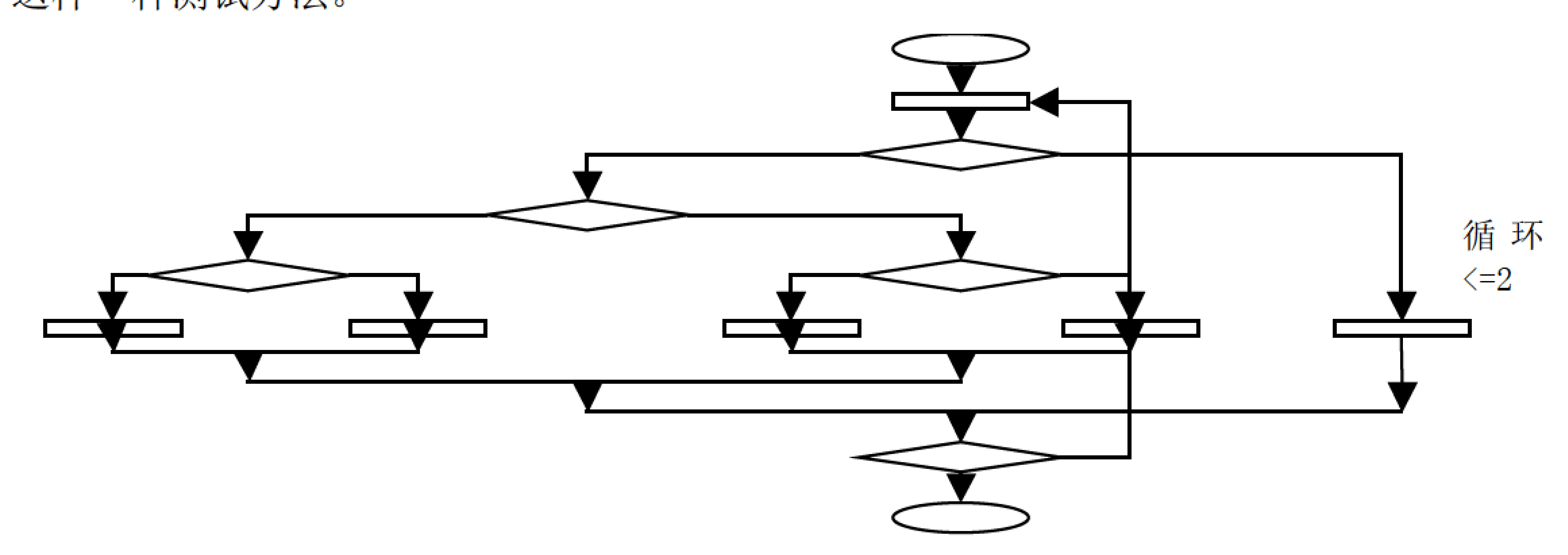


图 8-4　多次循环导致的天文路径数

基本路径测试是 McCabe 提出的一种白盒测试方法。使用这种方法设计测试用例时，首先要计算程序的圈复杂度，并以圈复杂度为指南定义执行路径的基本集合，从该基本集合导出的测试用例可以保证程序中的每条语句至少执行一次，而且每个条件在执行时都分别取真、假两种值。

使用基本路径测试方法设计测试用例的步骤如下：

1）根据过程设计结果画出相应的流图

程序流图是描述程序控制流的一种图示方法。其中，基本的控制结构对应的图形符号如图 8-5 所示。在图 8-5 中，符号○称为控制流图的一个节点，它表示一条或多条无分支的源程序语句。

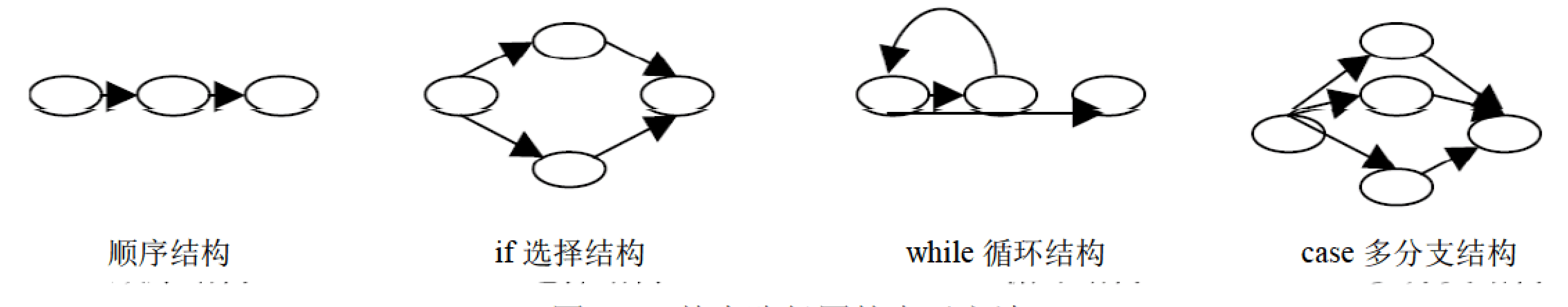


图 8-5　基本路径图的表示方法

例如，为了用基本路径测试方法测试下列用PDL语言描述的求平均值的过程，首先画出图8-6所示的流图。为了正确清楚地画出流图，我们对被映射为流图节点的语句编了序号。

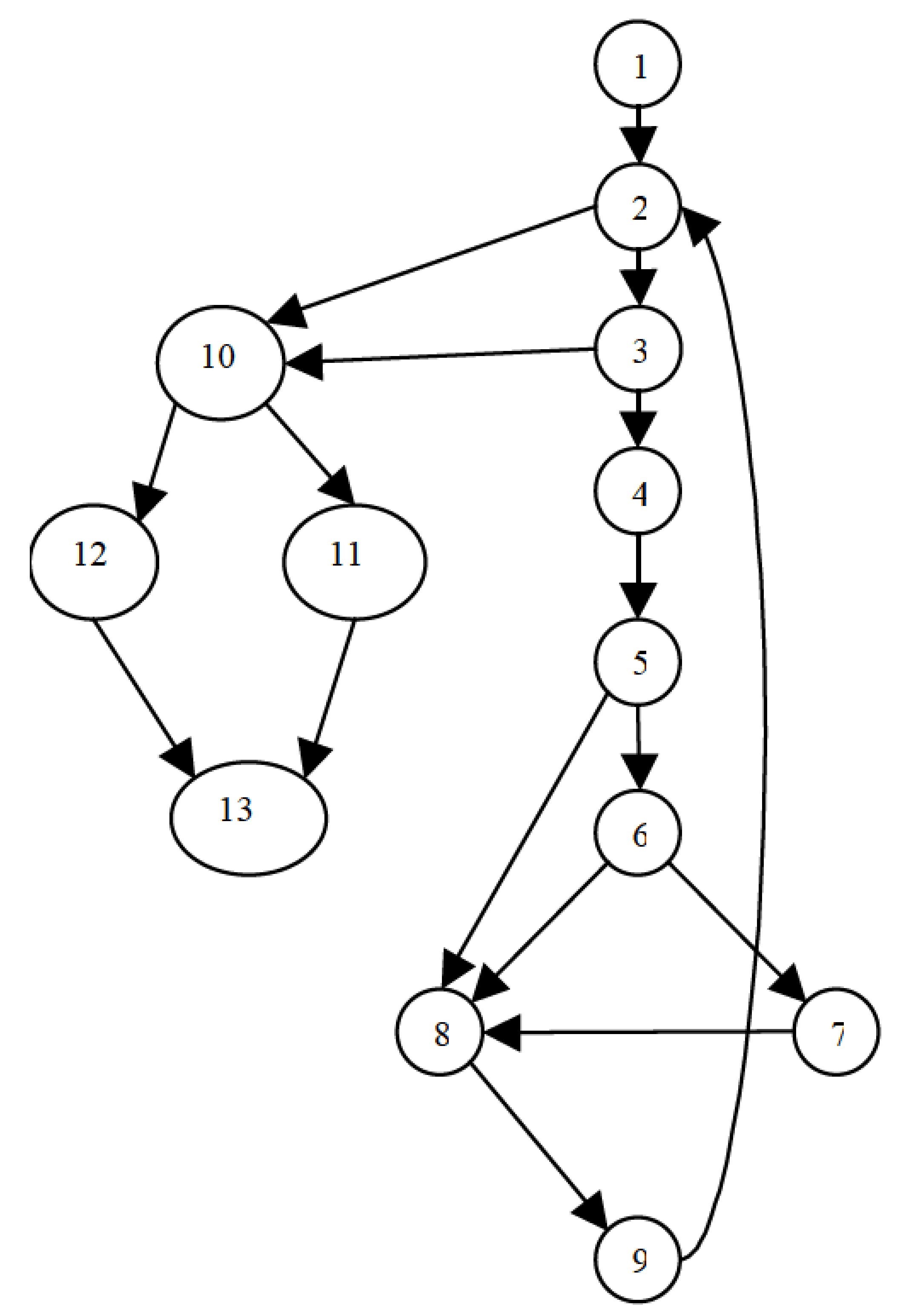


图 8-6　求平均值过程的流图

利用 McCabe 圈复杂度的计算公式 V(G)=m−n+2p(V(G))是强连通有向图 G 中的环数，m 是 G 中的弧数，n 是 G 中的节点数，p 是 G 中分离部分的数目。对于一个正常的程序来说，程序图总是连通的，即 p=1,V(G)=m−n+2，可进行基本路径计算。

```
PROCEDURE average;
/*计算不超过 100 个在规定值域内的有效数字的平均值；同时计算有效数字的总和及个数。*
 INTERFACE RETURNS average， total， input， valid;
 INTERFACE ACCEPTS value， minimum， maximum;
```

```
  TYPE value[1...100] IS SCALAR ARRAY;
  TYPE average,  total,  input,  total.valid;
  minimum,  maximum,  sum IS SCALAR;
  TYPE i IS INTEGER;

1: i=1;
   total.input = total.valid=0;
   sum = 0;
2: DO WHILE value[i] <> -999
3:     AND total.input < 100
4: increment total.input by 1;
5: IF value[i] >=minimum
6:     AND value[i]<=maximum
7: THEN increment total.valid by 1;
   sum = sum+value[i];
8: ENDIF
   increment i by 1;
9: ENDDO
10:IF total.valid >0
11:THEN average = sum/total.valid;
12:ELSE average = -999;

13:ENDIF
END average
```

圈复杂度用于度量程序的逻辑复杂性。有了描绘程序控制流的流图之后，可以计算出流图的圈复杂度。经计算，图 8-6 所示流图的圈复杂度是 6。

2) 确定线性独立路径的基本集合

所谓独立路径，是指至少引入程序的一个新处理语句集合或一个新条件的路径，用流图术语描述，独立路径至少包含一条在定义该路径之前不曾用过的边。

使用基本路径测试方法设计测试用例时，程序的圈复杂度决定了程序中独立路径的数量，而且这个数量是确保程序中所有语句至少被执行一次所需的测试数量的上界。

对于图 8-6 所描述的求平均值的过程来说，由于圈复杂度为 6，因此有 6 条独立路径。例如，下面列出了几条独立路径：

路径 1：1—2—10—11—13

路径 2：1—2—10—12—13

路径 3：1—2—3—10—11—13

路径 4：1—2—3—4—5—6—8—9—2—…

路径 5：1—2—3—4—5—6—7—8—9—2—…

路径 6：1—2—3—4—5—8—9—2—…

路径后面的省略号表示，可以后接通过控制结构其余部分的任意路径。

通常在设计测试用例时，识别出判定节点是非常重要的。本例中的节点 2、3、5、6 和 10 就是判定节点。

3) 设计可强制执行基本集合中每条路径的测试用例

选取测试数据，使得在测试每条路径时都适当地设置好各个判定结点的条件。例如，

可以测试上一步得出的基本集合的测试用例。

(1) 路径 1 的测试用例：

```
value[k]=有效输入值，其中 k<i (i 的定义在下面)
value[i]= −999，其中 2 <=i <=100
```

预期结果：基于 k 的正确平均值和总数。

其中，路径 1 无法独立测试，必须作为路径 4、5、6 的一部分来测试。

(2) 路径 2 的测试用例：

```
value[1]= −999
```

预期结果：average = −999，其他都保持初始值。

(3) 路径 3 的测试用例：

试图处理 101 个或更多个值。

前 100 个值应该是有效的输入值。

预期结果：前 100 个数的平均值，总数为 100。

其中，路径 3 同路径 1 一样，也无法单独测试，必须作为路径 4、5、6 的一部分来进行测试。

(4) 路径 4 的测试用例：

```
value[i]=有效输入值，其中 i<100
value[k]>maximum，其中 k<i
```

预期结果：基于 k 的正确平均值和总数。

在测试的过程中，执行每个测试用例并把实际输出结果与预期结果相比较。一旦执行完所有的测试用例，就可以确保程序中所有语句都被至少执行了一次，而且每个判定条件都分别取过 true 和 false 值。

3. 循环路径测试覆盖

循环路径测试分为：0 次循环(检查跳出循环)，1 次循环(检查循环初始值)，2 次循环(检查多次循环)，m 次循环(检查某次循环)，最大次数循环，比最大次数多一次、少一次循环，检查循环次数边界。

循环使路径数量急剧增长，为此我们要对循环过程进行简化：无论循环的形式和实际执行循环体的次数多少，只考虑循环一次和 0 次，即进入循环体一次和跳出循环体。

路径覆盖的利弊：①实现了所有路径的测试，发现错误能力强；②某些条件错误可能无法发现；③路径数庞大，不可能覆盖所有路径；④用例数量增加。

8.1.3 数据流测试

数据流测试最初是随着编译系统要生成有效的目标码而出现的，主要用于代码优化。现在主要为发现定义/引用异常缺陷，例如：变量被定义，但从来没有使用(引用)；所使用的变量没有被定义；变量在使用之前被定义两次。

数据流测试与数据流图之间并没有什么关系，数据流测试关注变量赋值与使用位置，它是一种结构性测试方法，可把它看作基于路径测试的一种改良方案(进行真实性检查)。

数据流测试一方面仍会用到路径测试的一些研究成果，另一方面也考虑向功能性测试目标靠近。

数据流测试重点关注的是变量的定义与使用测试。事实上，在调试修改 bug 时，我们也会经常这样做。例如在一段代码中搜索某个变量所有的定义、使用位置，思考在程序运行时该变量的值会如何变化，从而分析 bug 产生的原因。数据流测试将这种方法形式化，这样也便于构造算法，实现自动化分析。

数据流测试或定义/使用测试是以程序数据流的视角(程序是一个程序元素对数据访问的过程)，基于数据流关系(数据“定义-使用”对)，使用程序图来描述数据“定义-使用”对，通过对路径进行真实性检查来发现数据的不正确定义及使用问题，一种简单的数据流测试策略是要求覆盖每个定义-使用路径一次。

1. 定义-使用的相关定义

在下面的定义中，P 代表程序，G(P)为程序图，V 为变量集合，P 的所有路径集合为 PATH(P)。

节点 n 是变量 v 的定义节点，记做 DEF(v,n)。如果执行对应这种语句的节点，那么与该变量关联的存储单元的内容就会改变。输入语句、赋值语句、循环控制语句和过程调用，都是定义节点语句的例子。

节点 n 是变量 v 的使用节点，记做 USE(v,n)。如果执行对应这种语句的节点，那么与该变量关联的存储单元的内容会保持不变。语句、赋值语句、条件语句、循环控制语句和过程调用，都是使用节点语句的例子。

例 8-2

```
a=b；  DEF(1)={a}，  USES(1)={b}
a=a+b；  DEF(1)={a}，  USES(1)={a，b}
```

如果 USE(v,n)是谓词使用(在条件判断语句中)，则记做 P-use；如果 USE(v,n)是运算使用(在计算表达式中)，则记为 C-use。

对应于谓词使用的节点，永远有外度>=2；对应于计算使用的节点，永远有外度<=1。

注意：

内度(即有向图中节点的内度)是将该节点作为终止节点的不同边的条数，外度(即有向图中的外度)是将该节点作为开始节点的不同边的条数。

变量 v 的定义-使用路径记做 du-path，对于 PATH(P)中的某条路径，如果定义节点 DEF(v,m)为该路径的起始节点，使用节点 USE(v,n)作为该路径的终止节点，则该路径是 v 的定义-使用路径。

变量 v 的定义-清除路径(define-clear path)记做 dc-path，对于变量 v 的某个定义-使用路径，如果除了起始节点之外没有其他定义节点，则该路径是变量 v 的定义-清除路径。

定义-使用路径和定义-清除路径描述了变量 v 从被定义的点到值被使用的点的源语句的数据流。通常不是定义-清除的定义-使用路径，有可能存在问题。

结合程序流图，找出所有变量的定义-使用路径，考查测试用例对这些路径的覆盖程度，就可以作为衡量测试效果的参考。例 8-3 给出了变量定义和使用分析的实例，如图 8-7 及

表 8-1 所示。

例 8-3

```
1 a=5;   // 定义 a
2 While(C1) {
3 if (C2){
4       b=a*a; //定义 b，使用 a
5       a=a-1; //定义且使用 a
6 }
7 print(a);    //使用 a
8 print(b); }//使用 b
```

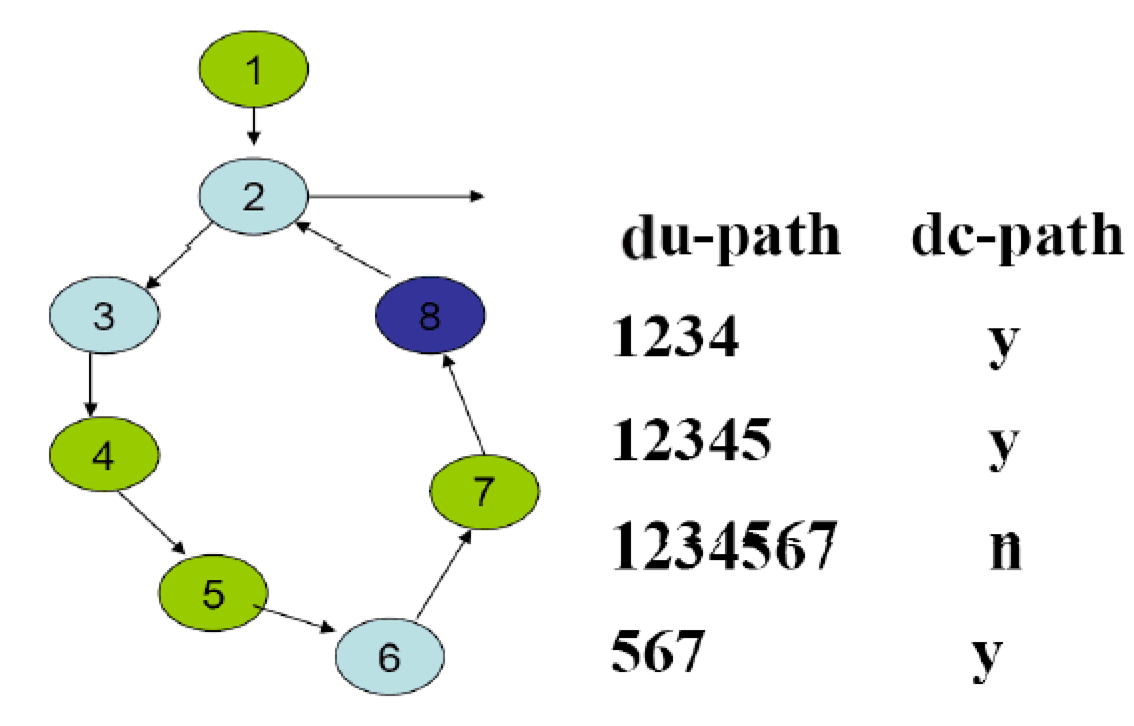


图 8-7　例 8-3 中的程序流图、定义-使用路径和定义-清除路径

表 8-1　例 8-3 中变量定义、使用及路径的情况列表

变量	定义节点	使用节点	路径(开始、结束)节点	是定义-清除吗？
a	1,5	4,5,7	1,4　1,5　1,7　5,7	是　否　否　是
b	4	8	4，8	是

2. 定义-使用路径的测试覆盖指标

对程序进行数据流分析的核心是定义一组叫做 Rapps-Weyuker 的数据流覆盖指标，即根据 Rapps 和 Weyuker 所定义的一组基于数据流的测试路径覆盖指标，结合程序流图，找出所有变量的定义-使用路径，考查测试用例对这些路径的覆盖程度，以作为衡量测试效果的参考。

数据流覆盖指标假设所有程序变量都标识了定义节点和使用节点，且关于各变量都表示了定义-使用路径。假定 T 为拥有变量集合 V 的程序 P 的程序图 G(P)中的一个路径集合。

1) 全定义(all-definition)覆盖准则

集合 T 满足程序 P 的全定义覆盖准则，当且仅当对于所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的一个使用的定义-清除路径，如图 8-8 所示。

2) 全使用(all-use)覆盖准则

集合 T 满足程序 P 的全使用覆盖准则，当且仅当对于所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的所有使用的定义-清除路径，如图 8-9 所示。

3) 全谓词使用(all predicate use)/部分计算使用(some calculation use)覆盖准则

集合 T 满足程序 P 的全谓词使用/部分计算使用准则，当且仅当所有变量 v∈V，T 包

含从 v 的每个定义节点到 v 的所有谓词使用的定义-清除路径，并且如果 v 的一个定义没有谓词使用，那么至少一个计算使用有一条定义-清除路径。

```
n1:  float X, Y, Z, U, W;
n2:  scanf("%f%f%f", &X, &Y, &Z);
n3:  U = (X − Y) ∗ 2;
n4:  if (X > Y)
n5:      W = U;
n6:  else W = Y;
n7:  if ((W + Z) > 100) {
n8:      X = X − 2;
n9:      Y = Y + W;
n10:     printf("Linear"); }
n11: else if (X ∗ X + Z ∗ Z >= 100) {
n12:     Y = X ∗ Z + 1;
n13:     printf("Nonlinear: Quadratic"); }
n14: if (U > 0)
n15:     printf("%f", U );
n16: else if ((Y − Sin(Z)) > 0)
n17:     printf("Nonlinear: Sine");
```

图 8-8　全定义覆盖示例

```
n1:  float X, Y, Z, U, W;
n2:  scanf("%f%f%f", &X, &Y, &Z);
n3:  U = (X − Y) ∗ 2;
n4:  if (X > Y)
n5:      W = U;
n6:  else W = Y;
n7:  if ((W + Z) > 100) {
n8:      X = X − 2;
n9:      Y = Y + W;
n10:     printf("Linear"); }
n11: else if (X ∗ X + Z ∗ Z >= 100) {
n12:     Y = X ∗ Z + 1;
n13:     printf("Nonlinear: Quadratic"); }
n14: if (U > 0)
n15:     printf("%f", U );
n16: else if ((Y − Sin(Z)) > 0)
n17:     printf("Nonlinear: Sine");
```

图 8-9　全使用覆盖示例

4) 全计算使用(all C-use)/部分谓词使用(some P-use)覆盖准则

集合 T 满足程序 P 的全计算使用/部分谓词使用准则，当且仅当所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的所有计算使用的定义-清除路径，并且如果 v 的一个定义没有计算使用，那么至少一个谓词使用有一条定义-清除路径。

5) 全定义-使用路径(all definition-use-paths，all-du-paths)覆盖准则

集合 T 满足程序 P 的全定义-使用路径准则，当且仅当所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的所有使用的定义-清除路径，并且这些路径要么有一次环经过，要么没有环路。

6) Rapps-Weyuker 数据流覆盖指标层次结构(见图 8-10)

在使用数据流覆盖指标时，需要注意全使用覆盖与全定义-使用覆盖的区别，如图8-11所示。

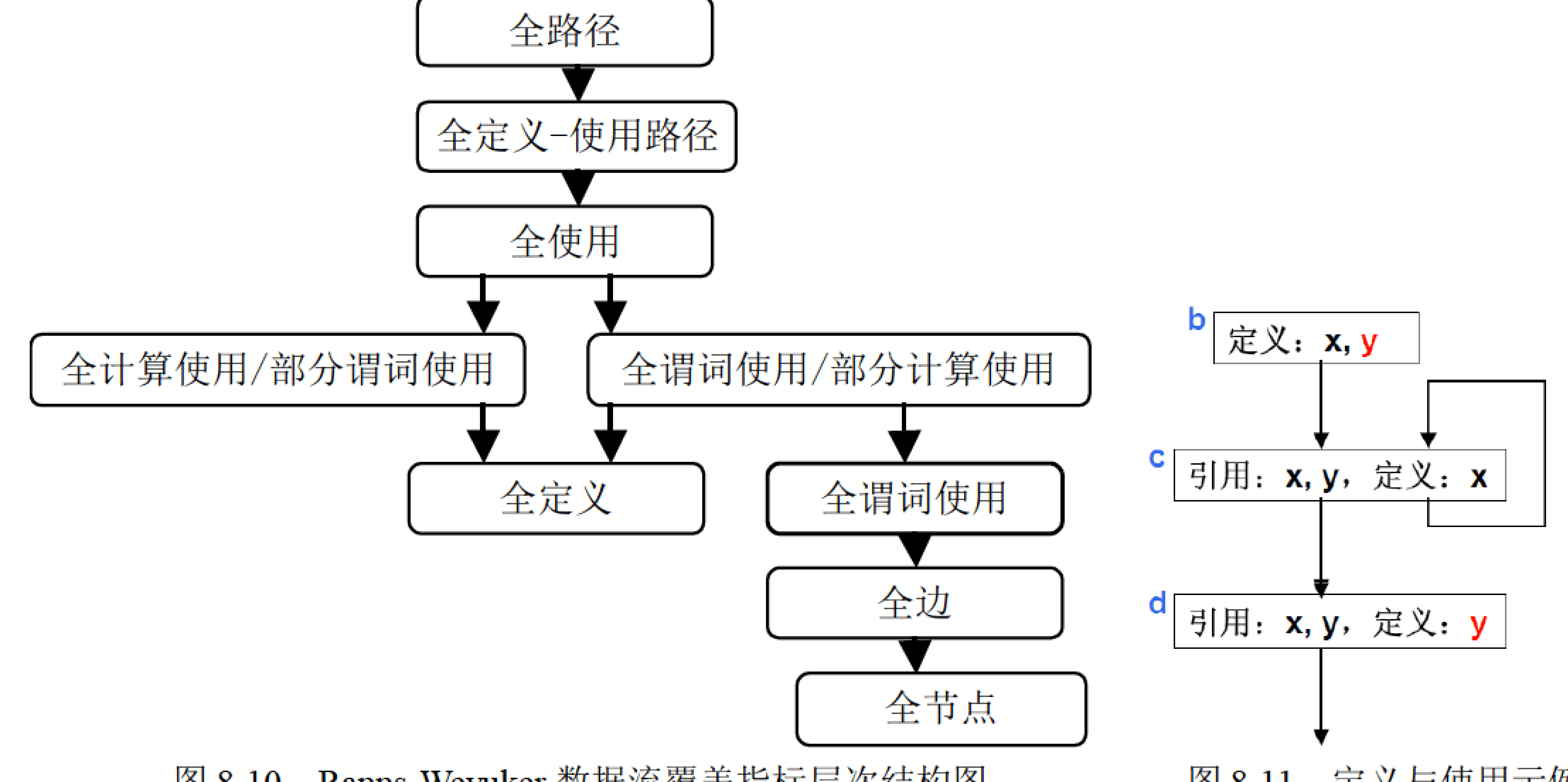


图 8-10　Rapps-Weyuker 数据流覆盖指标层次结构图　　图 8-11　定义与使用示例

(1) 节点 b 定义 y，节点 c 和 d 使用 b 定义的 y。

(2) 全使用覆盖虽然要求检查每个定义的所有可传递到的使用，但对于如何从一个定义传递到一个使用不作要求。

(3) 全定义-使用覆盖要求检查所有可能的路径，但为了避免有环路时的无穷多条路径，限制只检查无环路的或只包含一条环路的路径。

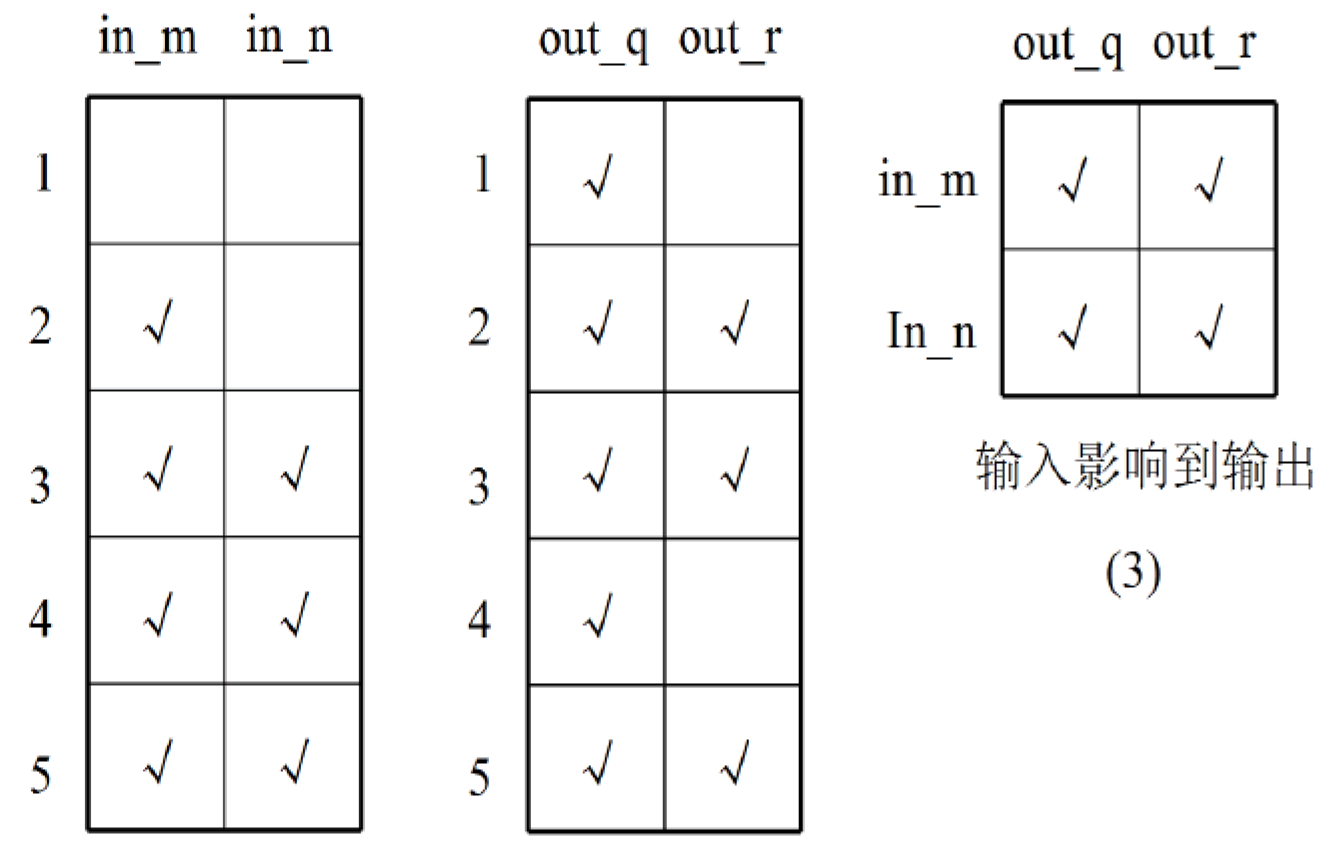
8.1.4　信息流分析

信息流分析是通过输入数据、输出数据、语句之间的关系(见图 8-12)来检查程序错误，还可用来分析是否存在无用的语句。下面我们通过例 8-4 来说明信息流分析的全过程。

信息流分析的具体作用有三点：①能够列出对输入变量的所有可能的引用；②在程序的任何指定点检查其执行可能影响某一输出变量值的语句；③输入/输出关系提供一种检查，看每个输出值是否有相关的输入值，而不是由其他值导出。

例 8-4　整除算法示例

```
输入：in_m 是被除数
      in_n 是除数
输出：out_q 是商
      out_r 是余数
      out_q = 0;
      out_r = in_m;
      while(out_r >= in_n)
      {
          out_q ++;
          out_r = out_r – in_n;
      }
```



	in_m	in_n
1		
2	√	
3	√	√
4	√	√
5	√	√

(1)输入变量影响语句

	out_q	out_r
1	√	
2	√	√
3	√	√
4	√	
5	√	√

(2)语句影响到输出变量

	out_q	out_r
in_m	√	√
In_n	√	√

输入影响到输出

(3)

图 8-12　数据、输出数据、语句之间的关系

8.1.5　覆盖率分析及测试覆盖准则

代码覆盖率是指采用白盒法进行测试时，考虑的是测试用例对程序内部逻辑的覆盖程度。最彻底的白盒法是覆盖程序中的每一条语句、每一个分支以及每一条路径，但这往往无法实现。因此，需要采用其他一些标准来度量覆盖的程度，并希望覆盖程度尽可能高些。

覆盖率分析主要对代码的执行路径覆盖范围进行评估，这些覆盖从不同要求出发，为设计测试用例提出依据。软件人员进行覆盖测试，主要想对程序模块进行如下几方面的检查：① 对程序模块的所有独立的执行路径至少测试一次；②对所有的逻辑判定，取真与取假的两种情况都至少测试一次；③在循环的边界和运行界限内执行循环体；④测试内部数据结构的有效性。

在覆盖测试实施过程中，我们需要注意如下几个问题：

(1) 内部动作是否按照规格说明书的规定正常进行。按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不用顾及它的功能。

(2) 覆盖测试的主要方法。覆盖测试的主要方法有逻辑驱动测试、基本路径测试等，主要用于软件验证。

(3) 覆盖法要求全面了解程序内部逻辑结构、对所有逻辑路径进行测试。覆盖法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。

(4) 不同软件的覆盖率分析对应的覆盖测试方法是可以有差别的。覆盖率方法有很多，取决于对软件的要求程度，比如航空、医疗软件要求严格，需要使用 DO-178B 的 MC/DC 覆盖率标准(判定/条件覆盖)。

(5) 覆盖测试只是根据程序的内部结构进行测试，而不考虑外部特性。如果程序结构本身有问题，比如程序逻辑有错误或是有遗漏，那是无法发现的。

1. 逻辑覆盖率计算方法

逻辑覆盖率主要是指语句覆盖率、判定覆盖率、条件覆盖率、判定/条件覆盖率、条件组合覆盖率和路径覆盖率。计算方法是：

覆盖率=(至少被执行一次的 item 数)/item 的总数

这个公式对 item 的覆盖情况进行计算，item 可以是需求、语句、分支、条件、路径等。覆盖率是用来度量测试完整性的一个手段，不是测试目的。通过覆盖率数据，我们可以知道我们的测试是否充分，测试的弱点在哪些方面，进而指导我们设计能够增加覆盖率的测试用例。

1) 语句覆盖率

语句覆盖率是指已执行的可执行语句占程序中可执行语句总数的百分比。计算公式如下：

语句覆盖率=(至少被执行一次的语句数量)/(可执行的语句总数)

注意：

对于 C 语言，可执行的语句不包括以#开头的#include、宏定义、预处理语句和注释语句。

复杂的程序不可能达到语句的完全覆盖，当然语句覆盖率越高越好。

通常，语句覆盖率可能出现这样的问题：语句覆盖率可能会很高，却有严重缺陷，如下面的示例所示。

```
if(x!=1)
{                                              测试用例
      statements;          }99 句              x = 2
      ……;                                      语句覆盖率99%
}                                              50%的分支没有达到
else
{
      statement;
}
```

1) 分支覆盖率

分支覆盖率是指已取过真和假两个值的判定占程序中所有条件判定个数的百分比。计算公式如下：

判定覆盖率=(判定结果至少被评价一次的数量)/(判定结果的总数)

2) 条件覆盖率

条件覆盖率是指在测试时，运行被测程序后所有判断语句中每个条件的可能取值(真值和假值)出现过的比率。计算公式如下：

条件覆盖率=(条件操作数值至少被评价一次的数量)/(条件操作数值的总数)

条件操作数是条件的具体取值(真或假)。

3) 分支-条件覆盖率或判断-条件覆盖率

在测试时，运行被测程序后，所有判断语句中每个条件的所有可能值(为真或为假)和每个判断本身的判定结果(为真或为假)出现的比率。计算公式如下：

分支-条件覆盖率=(条件操作数值或判定结果至少被评价一次的数量)/(条件操作数值总数+判定结果总数)

4) 路径覆盖率

在测试时，运行被测程序后，程序中所有可能的路径被执行过的比率。计算公式如下：

路径覆盖率=(至少被执行到一次的路径数)/(总的路径数)

注意：

N 个测试用例最多执行了 N 条路径。

分支覆盖可以保证语句覆盖，条件覆盖和分支覆盖不能互相保证，路径覆盖可以保证分支覆盖，路径覆盖不能保证条件覆盖。通常要求做到语句覆盖和分支覆盖；考虑到程序中错误(异常)处理工作的重要性及其结构相对简单，要求错误处理做到路径覆盖；对于质量要求高的软件，可根据情况提出条件覆盖、分支-条件覆盖以及路径覆盖要求。

2. 覆盖准则

测试覆盖准则目前常用到的主要有 ESTCA 和 LJSAJ。

1) FOSTER 的 ESTCA 覆盖准则

事实上覆盖率达到 100%是很难的，我们经常要借助我们的经验，有针对性地对容易发生问题的地方设计测试用例。

K.A.Foster 从测试工作实践的教训出发，吸收了计算机硬件的测试原理，提出了一种

经验型的测试覆盖准则，较好地解决了上述问题。

Foster 的经验型覆盖准则是从硬件的早期测试方法中得到启发。我们知道，在硬件测试中，对每一个门电路的输入、输出测试都是有额定标准的。通常，电路中一个门的错误常常是“输出总是 0”，或是“输出总是 1”。与硬件测试中的这一情况类似，我们常常要重视程序中谓词的取值，但实际上它可能比硬件测试更加复杂。Foster 通过大量的实验确定了程序中谓词最容易出错的部分，得出了一套 ESTCA (Error Sensitive Test Cases Analysis，错误敏感测试用例分析)规则。事实上，规则十分简单：

[规则 1]　对于 A rel B(rel 可以是 <、= 和 >)型的分支谓词，应适当地选择 A 与 B 的值，使得测试执行到该分支语句时，A < B\A = B 和 A > B 的情况分别出现一次。

[规则 2]　对于 A rel1 C(rel1 可以是 > 或 <，A 是变量，C 是常量)型的分支谓词，当 rel1 为 < 时，应适当地选择 A 的值，使 A = C − M(M 是距 C 最小的容器容许正数，若 A 和 C 均为整型，M = 1)。同样，当 rel1 为>时，应适当地选择 A，使 A = C + M。

[规则 3]　为外部输入变量赋值，使其在每一个测试用例中均有不同的值与符号，并与同一组测试用例中其他变量的值与符号不一致。

显然，上述[规则 1]是为了检测 rel 的错误，[规则 2]是为了检测差一之类的错误(比如本应是 IF A > 1 而错成 IF A > 0)，而[规则 3]则是为了检测程序语句中的错误(比如应引用一变量而错成引用一常量)。

上述三条规则并不是完备的，但在普通程序的测试中确是有效的。原因在于规则本身针对程序编写人员容易发生的错误，或是围绕发生错误的频繁区域，从而提高了发现错误的命中率。

2) Woodward 等人的层次 LCSAJ 覆盖准则

Woodward等人曾经指出结构覆盖的一些准则，如分支覆盖或路径覆盖，都不足以保证测试数据的有效性。为此，他们提出了一种层次LCSAJ覆盖准则。

LCSAJ 覆盖(Linear Code Sequence and Jump Coverage)即线性代码顺序和跳转覆盖，是 Woodward 等人提出来的一套覆盖率准则。LCSAJ 是一组顺序执行的代码，以控制流跳转为结束点。它的定义如下：

(1) 它起始于程序的入口或是一个可能导致控制流跳转的点。

(2) 它结束于程序的出口或是一个可能导致控制流跳转的点。

(3) 对于该点，跳转在后面的序列中产生。

它不同于 DD 路径。DD 路径是根据程序有向图决定的。一条 DD 路径是指两个判断之间的路径，但其中不再有判断。程序的入口、出口和分支节点都可以是判断点。而 LCSAJ 的起点是根据程序本身决定的。它的起点是程序第一行或转移语句的入口点，或是控制流可以跳转到达的点。因此，几个 LCSAJ 首尾相接可构成 LCSAL 串，组成程序的一条路径。第一个 LCSAJ 起点为程序起点，最后一个 LCSAJ 终点为程序终点。一条程序路径可能是由两个、三个或多个 LCSAJ 组成的。基于 LCSAJ 与路径的这一关系，Woodward 提出了 LCSAJ 覆盖准则，这是一种分层的覆盖准则：

第一层：语句覆盖。

第二层：分支覆盖。

第三层：LCSAJ 覆盖。即程序中的每一个 LCSAJ 都至少在测试中经历过一次。

第四层：两两 LCSAJ 覆盖。即程序中每两个首尾相连的 LCSAJ 组合在测试中都要经历一次。

第 n+2 层：每 n 个首尾相连的 LCSAJ 组合在测试中都要经历一次。

这说明，越是高层的覆盖准则越难满足。在实施测试时，若要实现上述 Woodward 层次 LCSAJ 覆盖，需要产生被测程序的所有 LCSAJ。

尽管 LCSAJ 覆盖比判定覆盖复杂得多，但是 LCSAJ 的自动化相对还是容易获得的。另外，对一个模块的微小变动可能对 LCSAJ 产生重大影响，因此维护 LCSAJ 的测试数据是相当困难的。大模块包含极其庞大的 LCSAJ，要获得 100%的覆盖率也是不现实的。然而 Woodward 等人提供的证据表明，把测试 100%的 LCSAJ 作为目标相对 100%的判定覆盖要有效得多。

8.2　黑盒测试

黑盒测试，又称为功能测试或数据驱动测试，是把测试对象当作看不见内部的黑盒。在完全不考虑程序内部结构和处理过程的情况下，测试者仅依据程序功能的需求规范考虑确定测试用例和推断测试结果的正确性。软件工程师使用黑盒测试可以导出执行程序所有功能需求的输入条件集，实现功能覆盖。功能覆盖中最常见的是需求覆盖，通过设计一定的测试用例，要求每个需求点都被测试到。因此，根据软件产品需求规格说明中的功能设计规格，在计算机上进行测试，以证实实现的每个功能是否符合要求。

黑盒测试意味着要在软件的接口处进行测试，它着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。因此，可以说黑盒测试是站在使用软件或程序的角度，从输入数据与输出数据的对应关系出发进行的测试，如图 8-13 所示。

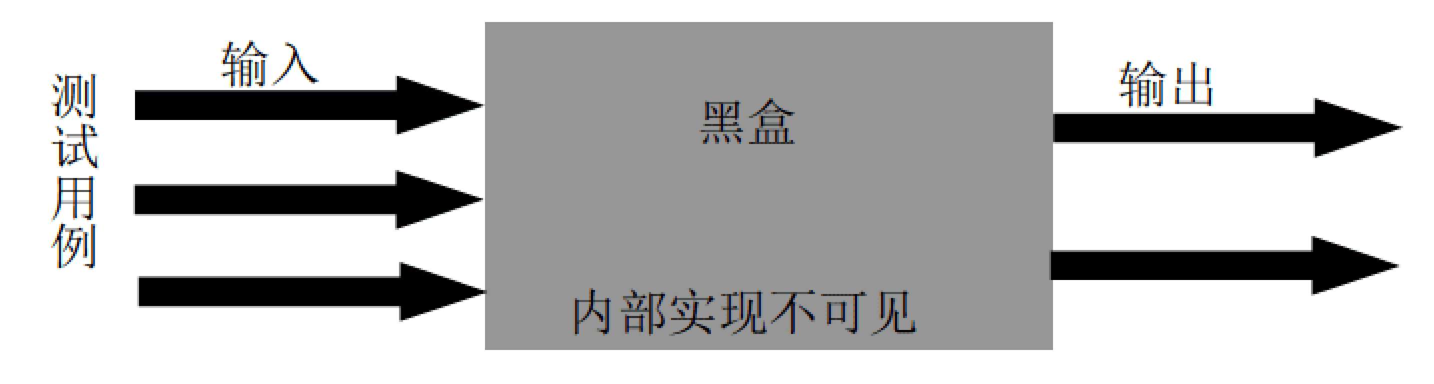


图 8-13　黑盒测试模型

很明显，如果外部特性本身设计有问题或规格说明中的规定有误，用黑盒测试方法是发现不了的。因此，黑盒测试不能替代白盒测试，而是用来发现白盒测试以外的其他类型的错误，比如：①功能不对或遗漏；②接口错误或界面错误；③数据结构或外部数据库访问错误；④性能错误；⑤初始化和中止错误。

黑盒测试直观的想法就是：既然程序被规定做某些事，那么我们就看看它是不是在任何情况下都做得对。换言之，要用黑盒测试发现程序中的错误，必须在所有可能的输入条件和输出条件下确定测试数据，检查程序是否都能产生正确的输出。很显然这是不可能的，因为穷举测试数量太大，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试，无法完成。

假设程序 P 有输入量 X 和 Y 及输出量 Z，在字长为 32 位的计算机上运行。若 X、Y 取整数，按黑盒方法进行穷举测试，可能采用的测试数据组个数为：$2^{32}\times2^{32}=2^{64}$。

如果测试一组数据需要 1 毫秒，一年工作 365×24 小时，完成所有测试需要耗时 5 亿年。

为此黑盒测试也有一套产生测试用例的方法，用以产生有限的测试用例而覆盖足够多的任何情况。由于黑盒测试不需要了解程序内部结构，因此许多高层的测试(如确认测试、系统测试、验收测试)都采用黑盒测试。

要用黑盒测试发现程序中的错误，必须在各种可能的输入条件和输出条件下确定测试数据，以检查程序是否都能产生正确的输出。黑盒测试所考虑的不是控制结构，而是关注于信息域。它主要回答以下几方面的问题：①如何测试功能的有效性？②如何测试系统行为和性能？③何种类型的输入会产生好的测试用例？④系统是否对特定的输入值特别敏感？⑤如何分隔数据类的边界？⑥系统能够承受何种数据率和数据量？

黑盒测试首先要求每个软件特性或功能必须被一个测试用例或认可的异常所覆盖。另外，要求构造数据类型和数据值的最小集测试，即用一系列真实的数据类型和数据值运行，测试超负荷、饱和及其他最坏情况的结果；用假想的数据类型和数据值运行，测试排斥不规则输入的能力。最后，对影响性能的关键模块(如基本算法)，应测试模块性能(包括精度、时间、容量等)。此时不仅要考核程序是否做了该做的，还要考核程序是否没做不该做的；同时还要考查程序在其他一些情况下是否正常，这些情况包括数据类型和数据值的异常，等等。

应用黑盒测试方法，所设计出的测试用例集都要满足以下两个标准：

(1) 所设计出的测试用例能够减少为达到合理测试所需要设计的测试用例的总数。

(2) 所设计出的测试用例能够告诉我们是否存在某些类型的错误，而不是仅仅指出与特定测试有关的错误是否存在。

黑盒测试只有测试通过和测试失败两种结果，在进行通过测试时，实际上是确认软件能做什么，而不会去考验其能力如何。测试人员只运用最简单、最直观的测试用例。在设计和执行测试用例时，总是先要进行通过测试，即在进行破坏性试验之前，看一看软件的基本功能是否能够实现。在确信软件正确运行之后，可采取各种手段通过搞垮软件来找出缺陷。

黑盒测试与软件如何实现无关，如果实现发生变化，黑盒测试用例仍然可用(可重用性，面向回归测试)。另外，黑盒测试用例开发可以与软件开发同时进行，这样可节省软件开发时间，通过软件的用例(use case)就可以设计出大部分黑盒测试用例。

当然，黑盒测试也存在一些问题：测试用例数量较大，测试用例可能产生很多冗余，而且功能性测试的覆盖范围不可能达到 100%。

黑盒测试主要用到的方法有：等价类划分法、因果图法、边值分析法、猜错法、随机数法等。这些测试方法是从更广泛的角度进行黑盒测试。每种方法都力图能涵盖更多的任何情况，但又各有长处。综合使用这些方法，会得到较好的测试用例集。

8.2.1　等价类划分

等价类划分法是典型的黑盒测试方法，用该方法设计测试用例时完全不必考虑软件结构，只需要考虑需求规格说明书中的功能要求。等价类划分法是把程序的输入域划分成若

干部分，然后从每个部分选取少数代表性数据当作测试用例。每一类代表性数据在测试中的作用等价于这一类中的其他值。也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误；反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误。使用这一方法设计测试用例，首先必须在分析需求规格说明的基础上划分等价类，列出等价类表。

使用等价类划分法设计测试方案时，首先要划分输入数据的等价类，为此需要研究程序的功能说明。等价类实际上就是某个输入域的一个子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。等价类的划分有两种不同的情况：有效等价类和无效等价类。有效等价类是指对于程序的规格说明来说，是由合理的、有意义的输入数据构成的集合；无效等价类是指对于程序的规格说明来说，是由不合理的、无意义的输入数据构成的集合。在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。

在确定输入数据的等价类时还常常需要分析输出数据的等价类，以便根据输出数据的等价类导出对应的输入数据等价类。

划分等价类需要经验，下面结合具体实例给出几条确定等价类的原则：

(1) 如果规定了输入条件的范围，可以划分出一个有效等价类和两个无效等价类。例如，在程序的规格说明中，对输入条件有如下规定：

"……入数值的范围是 1 至 999……

则有效等价类是 1<= 输入数值 <= 999，两个无效等价类是输入数值 < 1 和输入数值 > 999。

(2) 如果输入条件规定了输入值的集合，或是规定了必须如何的条件，这时可以确定一个有效等价类和一个无效等价类。例如，在某一种语言中规定了变量标识符是以字母打头的字符串，那么所有以字母打头的构成了有效等价类，而不在此集合内的构成了无效等价类。

(3) 如果输入条件是布尔值，那么可以确定一个有效等价类和一个无效等价类。

(4) 如果规定了输入数据的一组值，而且程序对不同输入值做不同的处理，则每个允许的输入值是一个有效等价类，此外还有一个无效等价类(任何一个不允许的输入值)。例如，在教师分房方案中规定对教授、副教授、讲师和助教分别计算分数，做相应的处理。因此可以确定 4 个有效等价类，分别为教授、副教授、讲师和助教，以及一个无效等价类，它是所有不符合以上身份的人员输入值的集合。

(5) 如果规定了输入数据必须遵守的规则，则可以确立一个有效等价类(符合规则)和若干个无效等价类。例如，在 C 语言中，处理时规定一条语句必须以分号；结束。这时，就可以确定一个有效等价类以"；"结束，若干个无效等价类以"，"结束、以"。"结束、以"："结束，等等。

(6) 如果确定知道已划分的等价类中各元素在程序中的处理方式不同，那么应将此等价类进一步划分成更小的等价类。

以上这些规则只是测试时可能遇到的情况中很小的一部分，实际的情况是千变万化的，根本不可能一一列出。为了能够正确地划分等价类，一是要注意经验积累，二是要正确分析被测程序的功能。此外，在划分无效等价类时，还要考虑编译程序的检错功能。最后再说明一点，上面那些规则虽然是针对输入数据的，但是对于输出数据也同样适用。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类，在图 8-14 中，在分出的等价类中按以下原则选择测试用例：

(1) 为每一个等价类规定一个唯一编号。

(2) 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止。

(3) 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

输入条件	有效等价类	无效等价类等价类
……	……	……
……	……	……

图 8-14　等价类表

下面用等价类划分法设计一个简单程序的测试方案。

假设有一个把数字串转换为整数的函数。运行程序的计算机字长 16 位，用二进制补码表示整数。这个函数是用 C 语言编写的，声明如下：

```
int strtoint(char shortstr[])
```

其中，参数 shortstr[]的定义如下：

```
char shortstr[6];
```

被处理的数字串是右对齐的，也就是说，如果数字串比 6 个字符短，就在它的左边补空格。如果字符串是负的，那么负号和最高位数字相邻。因为编译程序固有的检错功能，测试时不需要使用长度不等于 6 的数组作为输入参数，更不需要使用任何非字符数组类型的输入参数。通过分析这个程序的规格说明，可以划分出如下等价类：

有效输入的等价类：

(1) 由 1 到 6 个数字字符组成的数字串(最高位数字不是零)。

(2) 最高位数字是零的数字串。

(3) 最高位数字左邻是负号的数字串。

无效输入的等价类：

(1) 空字符串(全是空格)。

(2) 左边填充的字符既不是零，也不是空格。

(3) 最高位数字的右边由数字和空格混合组成。

(4) 最高位数字的右边由数字和其他字符混合组成。

(5) 负号与最高位数字之间有空格。

合法输出的等价类：

(1) 在计算机能表示的最小负整数和零之间的负整数。

(2) 零。

(3) 在零和计算机能表示的最大正整数之间的正整数。

非法输出的等价类：

(1) 比计算机能表示的最小负整数还小的负整数。

(2) 比计算机能表示的最大正整数还大的正整数。

根据上面划分出 x 的等价类，可以设计出如下测试用例：

(1) 由 1 到 6 个数字组成的数字串，输出是合法的正整数。

输入：'　　1'；预期的输出：1。

(2) 最高位数字是零的数字串，输出是合法的正整数。

输入：'000001'；预期的输出：1。

(3) 负号与最高位数字相邻，输出合法的负整数。

输入：'-00001'；预期的输出：-1。

(4) 高位数字是零，输出是零。

输入：'000000'；预期的输出：0。

(5) 太小的负整数。

输入：'-47561'；预期的输出：错误——无效输入。

(6) 太大的正整数。

输入：134567；预期的输出：错误——无效输入。

(7) 空字符串。

输入：'　　'；预期的输出：错误——没有数字。

(8) 字符串左边字符既不是零，也不是空格。

输入：'#####1'；预期的输出：错误——填充错误。

(9) 最高位数字的后面有空格。

输入：' 1 2'；预期的输出：错误——无效填充。

(10) 最高位数字的后面有其他字符。

输入：' 1##2'；预期的输出：错误——无效输入。

(11) 负号和最高位数字之间有空格。

输入：' - 23'；预期的输出：错误——负号位置错误。

8.2.2　边界值分析

边界值分析是一种补充等价划分法的测试用例设计方法，它不是选择等价类的任意元素，而是选择等价类边界的测试用例。实践证明，采用边界值分析法设计测试用例进行软件测试，常常可以查出更多的错误，取得良好的测试效果。因为大量的错误发生在输入域或输出域的边界，而不是在其集合范围内。比如，在做三角形计算时，要输入三角形的三个边长：A、B 和 C。 我们应注意到这三个数值需要满足 A＞0、B＞0、C＞0、A＋B＞C、A＋C＞B、B＋C＞A，才能构成三角形。如果把六个不等式中的任何一个大于号＞错写成大于等于号≥，那就不能构成三角形。问题恰好出现在容易被疏忽的边界附近。

应用边界值分析法设计测试用例的原则就是确定输入或输出在边界的取值，如输入/输出域的最大值或最小值、第一个值或最后一个值、最大个数或最小个数以及刚刚超出边界的值。

使用边界值分析方法设计测试方案时，首先应该确定边界情况，这需要经验和创造性，通常输入等价类和输出等价类的边界，就是应该着重测试的程序边界情况。选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。下面是使用边界值分析方法选择测试用例的几条原则：

1) 如果输入条件规定了值的范围，那么应取刚达到这个范围的边界值，以及刚刚超越这个范围的边界值作为测试的输入数据

例如，输入值的范围是 1~9，则可以选取 1、9、0.9、9.1 作为测试输入数据。

2) 如果输入条件规定了输入值的个数，则用最大个数、最小个数、比最大个数大一个、比最小个数小一个的数作为测试数据

例如输入文件可以有 1 至 255 条记录，那么可以分别设计含有 1 条记录、255 条记录、0 条记录和 256 条记录的输入文件。

3) 根据规格说明的每个输出条件，使用原则 1。

例如，某程序的功能是计算折扣量，最低折扣是 0 元，最高折扣是 1000 元。设计一些测试用例，使它们刚好产生 0 元和 1000 元的结果。

4) 如果程序的规格说明给出的输入域或输出域是有序集合，那么应选取集合的第一个元素和最后一个元素作为测试用例

5) 分析规格说明，找出其他可能的边界条件

通常设计测试用例时总是将等价划分和边界值分析两种技术同时使用，例如为了测试前面所述的把数字串转换为整数的程序，除了 8.2.1 节中使用的价类划分方法设计出的测试用例外，还应该用边界值分析法加以补充。

a. 使输出刚好等于最小的负整数。

输入：'−32768'；预期的输出：−32768。

b. 使输出刚好等于最大的正整数。

输入：' 32767'；预期的输出：32767。

原来用等价类划分法设计出来的测试用例(5)最好改为：

c. 使输出刚刚小于最小的负整数。

输入：' −32769'；预期的输出：错误——无效的输入。

原来的测试用例(6)最好改为：

d. 使输出刚刚大于最大的正整数。

输入：' 32768'；预期的输出：错误——无效输入。

等价分类法与边界值分析法相比，等价分类法的测试数据是在各个等价类允许的值域内任意选取的，而边界值分析法的测试数据必须在边界值附近选取。一般来说，用边界值分析法设计的测试用例要比等价分类法的代表性更广，发现错误的能力也更强。但边界值分析法对边界的分析与确定比较复杂，要求测试人员有更多的经验和耐心。

8.2.3　因果图

前面介绍的等价类划分法和边界值分析法，都着重考虑输入条件，但未考虑输入条件

之间的关系。因果图(Cause-Effect Graphics)法充分考虑了输入情况的各种组合及输入条件之间的相互制约关系。因而，该方法能够帮助我们按一定步骤，高效率地选择测试用例，同时还能指出程序规格说明描述中存在着什么问题。

用因果图生成测试用例的基本步骤如下所述：

(1) 分析软件规格说明描述中，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，并给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明书描述中的语义，找出原因与结果之间，原因与原因之间对应的关系，然后根据这些关系，画出因果图。

(3) 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

(4) 把因果图转换成判定表。

(5) 把判定表的每一列拿出来作为依据，设计测试用例。

下面介绍一下因果图中出现的基本符号。

通常在因果图中用 Ci 表示原因，用 Ei 表示结果，基本符号如图 8-15 所示。各节点表示状态，可取值 1 或 0。1 表示某状态出现，0 表示某状态不出现。

主要原因和结果之间的关系如下：

(a) 恒等：表示原因与结果之间一对一的对应关系。若原因出现，则结果出现；若原因不出现，则结果也不出现。

(b) 非：表示原因与结果之间的一种否定关系。若原因出现，则结果不出现；若原因不出现，则结果出现。

(c) 或(V)：表示几个原因中若有一个出现，则结果出现；只有当这几个原因都不出现时，结果才不出现。

(d) 与(∧)：表示几个原因若都出现，结果才会出现。若几个原因中有一个不出现，结果就不会出现。

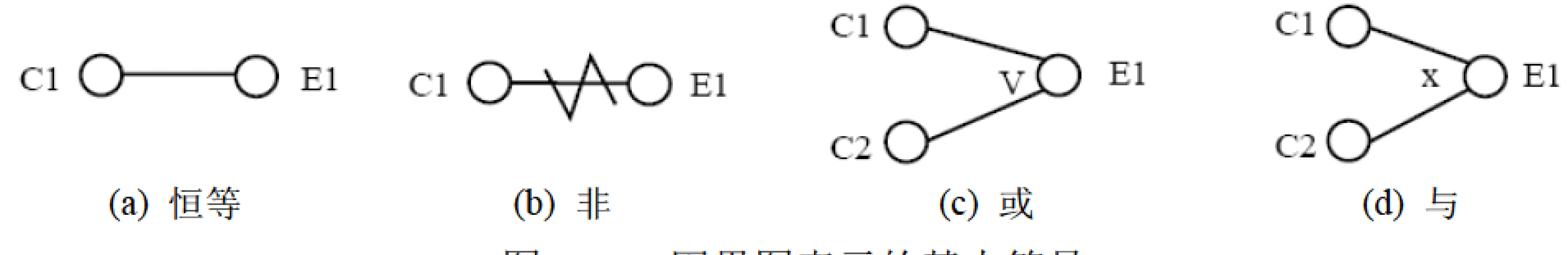


图 8-15　因果图表示的基本符号

下面是因果图中表示约束条件的符号。

为了表示原因与原因之间、结果与结果之间有可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。若从原因考虑，有以下 4 种约束，如图 8-16 所示。

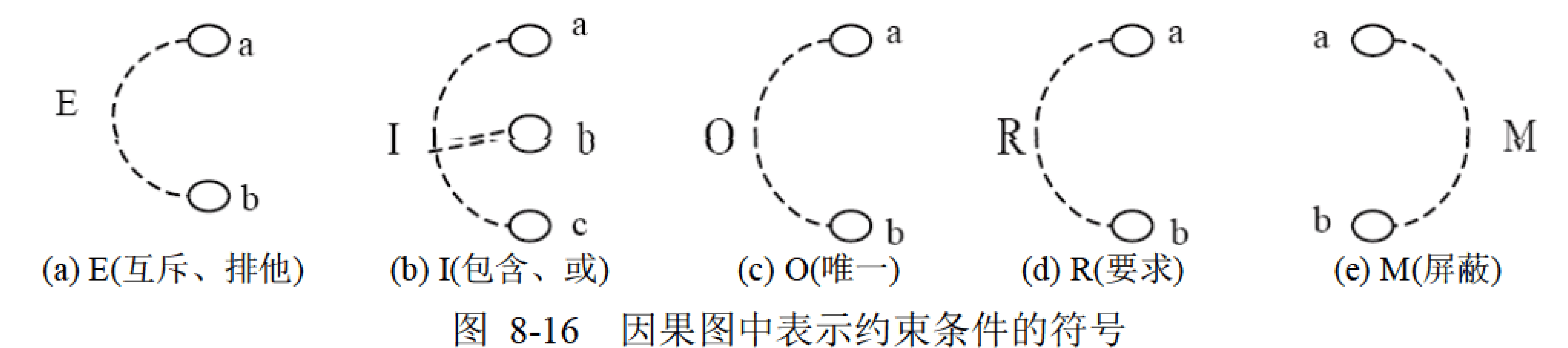


图 8-16　因果图中表示约束条件的符号

(a) E(互斥)：表示 a、b 两个原因不会同时成立，两个中最多有一个可能成立。

(b) I(包含)：表示 a、b、c 三个原因中至少有一个必须成立。

(c) O(唯一)：表示 a 和 b 中必须有一个且仅有一个成立。

(d) R(要求)：表示当 a 出现时，b 也必须出现；不可能 a 出现而 b 不出现。

若从结果考虑，还有一种约束：

(e) M(屏蔽)：表示当 a 是 1 时，b 必须是 0；而 a 为 0 时，b 的值不确定。

采用此方法时，首先分析规格说明书，确定原因和结果并把原因和结果连接成因果图。然后根据实际情况在因果图上使用若干特殊符号标明约束条件，最后把因果图转换成判定表，把判定表中每列的情况写成测试用例。

(1) 以自动售货机为例，列出原因和结果。

原因：①售货机有零钱找；②投入 1 元硬币；③投入 5 角硬币；④按下橙汁按钮；⑤按下啤酒按钮。

结果：㉑售货机“零钱找完”灯亮；㉒退还 1 元硬币；㉓退还 5 角硬币；㉔送出橙汁饮料；㉕送出啤酒饮料。

建立中间节点，表示处理过程的中间状态：⑪投入 1 元硬币且按下“饮料按钮；⑫按下橙汁或啤酒按钮；⑬应当找 5 角零钱并且售货机有零钱找；⑭钱已付清。

(2) 画出因果图，如图 8-17 示。所有原因节点列在左边，所有结果节点列在右边。

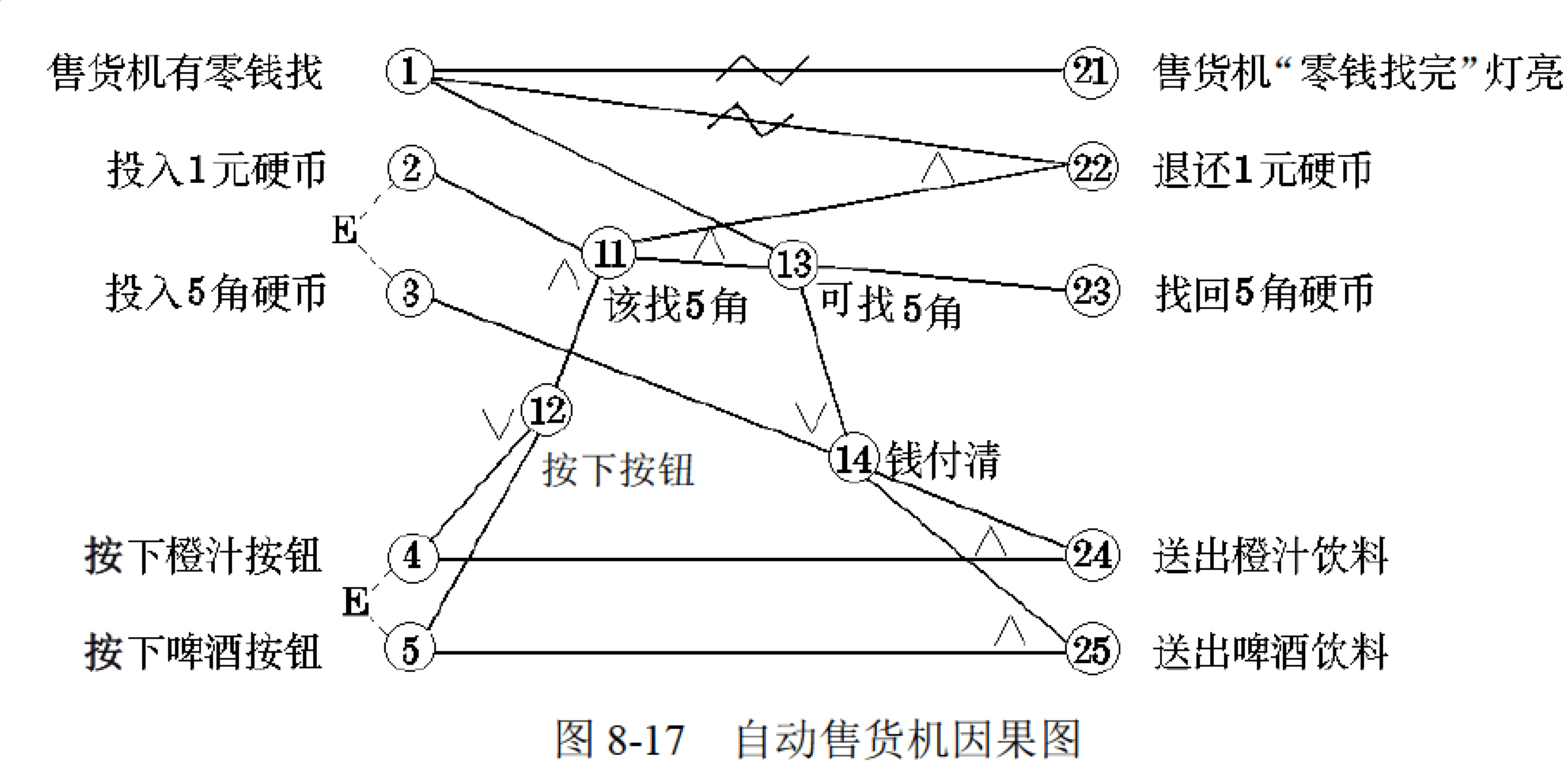


图 8-17　自动售货机因果图

(3) 由于②与③、④与⑤不能同时发生，因此分别加上约束条件 E。

(4) 将因果图转换为判定表，如图 8-18 示。判定表中没有被划去的一列就是一个测试用例。

判定表中阴影部分表示因违反约束条件的不可能出现的情况，删去；第 16 列与第 32 列因什么动作也没做，也删去；最后可把剩下的 16 列作为确定测试用例的依据。

总结：因果图法是一种非常有效的黑盒测试方法，它能够生成没有重复性的且发现错误能力强的测试用例，而且对输入、输出同时进行分析；从因果图生成的测试用例包括所有输入数据取真与取假的情况，构成的测试用例数目达到最少，且测试用例数目随输入数据数目的增加而线性增加；如果哪个开发项目在设计阶段就采用了判定表，也就不必再画因果图，而是可以直接利用判定表设计测试用例。

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

图 8-18　对应自动售货机因果图的判定表

8.2.4　随机测试

随机测试是指测试输入数据是从所有可能输入值中随机选取的，是一种基本的黑盒测试方法。随机选取用随机模拟的方法，包括用伪随机数发生器、硬件随机模拟器产生输入数据。这种方法能够获得大量的测试数据，测试人员只需要规定输入变量的取值区间，在需要的时候提供必要的变换机制，使产生的随机数服从预期的概率分布。

关于随机测试数据的选取方式，Laemmel 经分析认为，在测试次数很大时，可在数据输入空间中按均匀分布选用，在测试次数较少时，最好从常用的输入数据域以及最可能发生错误的输入数据域选用。随机测试与前面介绍的其他方法一起使用效果更佳。

但是随机测试不能事先将测试的输入数据存入文档，在排错时无法重现测试中错误发生的过程，也无法进行回归测试。补救的办法是将随机产生的测试数据记录备用。

8.2.5　猜错法

使用边界值分析法和等价类划分法，有助于设计出具有代表性的、容易暴露程序错误的测试方案。但是，不同类型、不同特点的程序通常又有一些特殊的容易出错的情况。此外，有时分别使用每组测试用例时程序都能正常使用，这些输入数据的组合却有可能检验出程序中存在的错误。一般来说，即使是一个比较小的程序，可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最有可能引起程序出错的方案。猜错法就是这样一种黑盒测试方法。

猜错法是基于经验和直觉推测程序中所有可能存在的各种错误，从而有针对性地设计测试用例的方法。猜错法的基本思想是：列举出程序中所有可能的错误和容易发生错误的特殊情况，根据它们选择测试用例。经过大量的经验总结，程序中容易出错的情况有：输入数据为零或输出数据为零，往往容易发生错误；如果输入或输出的数目允许变化，那么输入或输出的数目为 1 或 0 的情况是容易出错的情况等。除此之外，还要仔细分析程序的规格说明书，注意找出其中遗漏或省略的部分，以便设计出相应的测试方案，检测程序员对这些部分的处理是否正确。

此外，经验表明，在一段程序中已经发现的错误数目往往和尚未发现的错误数目成正比。例如，在 IBM OS/370 操作系统中，用户发现的全部错误的 47%只与操作系统 4%的模块有关。因此，在进一步测试时要着重测试那些已经发现较多错误的程序段。

8.3　测试用例设计

Grenford J. Myers 在 *The Art of Software Testing* 一书中提出：好的测试用例是指很可能找到迄今为止尚未发现的错误的测试，由此可见测试用例设计工作在整个测试过程中的地位。我们不能只凭借一些主观或直观的想法来设计测试用例，应该以一些比较成熟的测试用例设计方法为指导，再加上设计人员个人的经验积累来设计测试用例，两者相结合应该是非常完美的组合。

8.3.1　测试用例设计概念

测试用例目前没有经典的定义，比较通常的说法是：测试用例是为特定的目的而设计的一组测试输入、执行条件和预期结果，体现测试方案、方法、技术和策略，内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。测试用例是执行的最小实体。简单地说，测试用例就是设计一种场景，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。

随着中国软件业的日益壮大和逐步走向成熟，软件测试也在不断发展。其中，测试用例的设计和编制是软件测试活动中最重要的，是测试工作的指导，是软件测试必须遵守的准则，更是软件测试质量稳定的根本保障。

1. 高质量测试用例所应具备的特点

1) 正确性

正确性是对测试用例最基本的要求，测试用例最好要求输入用户实际数据以验证系统是否满足需求规格说明书的需求，并且测试用例中的测试点应保证至少覆盖需求规格说明书中的各项功能。

2) 完整性

完整性同样是对测试用例最基本的要求，尤其是一些基本功能，如有遗漏，那是不可原谅的。另外，完整性还体现在临界测试、压力测试、性能测试等方面，这方面测试用例也要能够涉及。

3) 准确性

按测试用例的输入实施测试后，要能够根据测试用例描述的输出得出正确的结论，不能出现模糊不清的语言。

4) 清晰、简洁

好的测试用例描述清晰，每一步都应有相应的作用，有很强的针对性，不应出现一些冗繁无用的操作步骤。测试用例不应太简单，也不能太过复杂，最大操作步骤最好控制在

15 步之内。

5) 可维护性

由于软件开发过程中需求变更等原因的影响，常常需要对测试用例进行修改、增加、删除等，以便测试用例符合相应测试要求。测试用例应具备这方面的功能。

6) 适应性

测试用例应该适合特定的测试环境以及符合整个团队的测试水平，如纯英语环境下的测试用例最好使用英文编写。

7) 可重用性

要求不同测试者在同样测试环境下使用同样测试用例都能得出相同结论。

8) 其他

如可追溯性、可移植性也是对编写测试用例的基本要求。另外，好的测试用例也是最有可能抓住错误的；还有，测试用例不要是重复的或多余的；最后，希望是一组相似测试用例中最有效的。

2. 测试用例设计基本原则

设计测试用例时，应遵循以下一般性原则：

1) 基于测试需求的原则

应按照测试类别的不同要求设计测试用例。例如，单元测试依据详细设计说明，集成测试依据概要设计说明，配置项测试依据软件需求规格说明，系统测试依据用户需求(系统/子系统设计说明、软件开发计划等)。

2) 基于测试方法的原则

应明确所采用的测试用例设计方法，为达到不同的测试充分性要求，应采用相应的测试方法，如等价类划分、边界值分析、猜错法、因果图等方法。

3) 兼顾测试充分性和效率的原则

测试用例集应兼顾测试的充分性和测试的效率；每个测试用例的内容也应完整，具有可操作性。

4) 测试用例的代表性

能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

5) 测试结果的可判定性

即测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

6) 测试执行的可再现性

即对同样的测试用例，系统的执行结果应当是相同的，也就是说是可再现的。

在此基础上，可提出更细或更具体的设计原则：

(1) 一个测试用例对应一个功能点，每个测试用例都要有测试点，找准一个测试点则可，不能同时覆盖很多功能点，否则执行起来牵连太大，不易检查、维护和管理。

(2) 测试用例要易读，要从执行者的角度去写测试用例，最好不要有太多的术语在里面，要指明具体位置。

(3) 测试用例的执行粒度越小越好。因为只有这样，测试用例所覆盖的边界定义才会更加清晰，测试结果对产品问题的指向性更强，测试用例间的耦合度最低，彼此之间的干扰也就越低。这带来的直接好处是：测试用例的调试、分析和维护成本最低。

(4) 步骤要清晰。一个测试用例有多个步骤，但要有重点。有步骤指明人们怎么去操作。

(5) 结果要明确。期望结果要清晰地指明一个操作之后应该看到什么结果(最好不要用正确、正常或错误之类的含糊主观的字眼)。良好的测试用例一般会有两种状态：通过(Pass)或未通过(Failed)以及未进行测试(Not Done)。如果测试未通过，一般会有测试的错误(bug)报告进行关联；如果未进行测试，则需要说明原因(测试用例本身的错误、测试用例目前不适用、环境因素等)。

(6) 尽量将具有类似功能的测试用例抽象并归类。这主要是因为软件测试过程是无法进行穷举测试的，因此，对类似测试用例的抽象过程显得尤为重要，一个好的测试用例应该足以代表一组或一系列的测试过程。

(7) 尽量避免冗长和复杂的测试用例。这么做为的是在测试执行过程中确保测试用例输出状态及验证结果的唯一性，从而便于跟踪和管理。

(8) 总体设计思路是先进行基本功能测试，再进行复杂功能测试；先进行一般用户使用测试，再进行特殊用户使用测试；先进行正常情况的测试，再进行异常情况(内存和硬件的冲突、内存泄漏、破坏性测试等)的测试，这样可以先易后难，循序渐进，确保正常情况下基本功能能够走通。

(9) 试着用测试用例替代产品文档。由于很多软件开发文档更新不及时，或者没有真正反映出产品的变化，因此也就无法准确地反映出产品功能当前的状态，而最终真正能够一直准确、有效地描述产品功能的只有产品代码和测试用例。产品代码实现了产品功能，并且一定准确描述了产品的当前功能。另外，测试也应该忠实反映产品功能，否则测试用例就会执行失败。因此，可把对测试用例的理解上升到一个新的高度，让它扮演产品描述文档的功能。这就要求我们编写的测试用例足够详细，测试用例的组织要有条理、分主次，并用测试用例管理工具来支撑。

(10) 测试用例设计要考虑单次投入成本和多次使用成本。因为编写测试用例一般是在测试的计划阶段进行的，虽然后期会有小的改动，但绝大多数是在一开始的设计阶段就基本成型了；自动化测试用例也是如此，它也属于一次性投入；测试用例(包括手工和自动化测试用例)的执行则是多次投入成本，因为每一个新版本建立时都要执行所有的测试用例、分析测试结果、确定测试失败的原因，以验证该版本整体质量是否达到指定的标准。

3. 测试用例覆盖内容

测试用例要测试被测软件所有的功能，并且希望选用少量、高效的测试数据进行尽可能完备的测试。测试用例应覆盖以下几个方面：

(1) 正确性测试。输入用户实际数据以验证系统是否满足需求规格说明书中的要求；测试用例中的测试点应首先保证至少覆盖需求规格说明书中的各项功能，并且正常。

(2) 容错性(健壮性)测试。程序能够接收正确数据输入并且产生正确(预期)的输出，输

入非法数据(非法类型、不符合要求的数据、溢出数据等)，程序应能给出提示并进行相应处理。把自己想象成一名对产品操作一点也不懂的客户在进行任意操作。

(3) 完整(安全)性测试。对未经授权的人使用软件系统或数据的企图，系统能够控制的程度，程序的数据处理能够保持外部信息(数据库或文件)的完整。

(4) 接口间测试。测试各个模块相互间的协调和通信情况，数据输入/输出的一致性和正确性。

(5) 数据库测试。依据数据库设计规范对软件系统的数据库结构、数据表及其之间的数据调用关系进行测试。

(6) 边界值分析法。确定边界情况(刚好等于、稍小于、稍大于和刚刚大于等价类边界值)，针对被测系统在测试过程中主要在边界值附近选取输入的一些合法数据/非法数据。

(7) 压力测试。输入 10 条记录运行各个功能，输入 30 条记录运行，输入 50 条记录运行……进行测试。

(8) 等价划分。将所有可能的输入数据(有效的和无效的)划分成若干个等价类。

(9) 错误推测。主要是根据测试经验和直觉，参照以往的软件系统出现错误之处。

(10) 效率。完成预定的功能，系统的运行时间(主要针对数据库而言) 。

(11) 可理解(操作)性。理解和使用该系统的难易程度(界面友好性)。

(12) 可移植性。在不同操作系统及硬件配置情况下的运行性。

(13) 回归测试。按照测试用例将所有的测试点测试完毕，测试中发现的问题开发人员已经解决，进行下一轮的测试。

(14) 比较测试。将已经发版的类似产品或原有的老产品与测试的产品同时运行比较，或与以往的测试结果比较。

说明：针对不同的测试类型和测试阶段，测试用例编写的侧重点有所不同。

- 其中(1)、(2)、(6)、(8)、(9)、(13)为模块(组件、控件)测试、组合(集成)测试、系统测试都涉及的并且要重点进行测试。
- 单元(模块)测试、组件及控件测试要重点测试(5)。
- 组合(集成)测试重点进行接口间数据输入及逻辑的测试，即(4)。
- 系统测试重点测试(3)、(7)、(10)、(11)、(12)、(14)。
- 对于压力测试和可移植性测试，如果是公司的系列产品，选用其中有代表性的产品进行一次代表性测试即可。
- 在基础的功能测试用例设计完成后，其他的测试项目只编写设计与之不同部分的测试用例。
- 每个测试项目的测试用例不是一成不变的，随着测试经验的积累或在测试其他项目中发现有测试不充分的测试点时，可以不断地补充完善测试项目的测试用例。

8.4.2　测试用例编写要素与模板

测试用例设计是测试工作的核心任务之一，也是工作量最大的任务之一，编写良好的测试用例能够较好地提高测试用例的设计质量，方便跟踪测试用例的执行结果，自动生成测试用例的覆盖率报告。一般来说，编写测试用例所涉及的内容或要素，以及样式均大同小异，一般都包含主题、前置条件、执行步骤、期望结果等。测试用例可以用数据库、Word、

Excel、XML 等格式进行存储和管理，市面上亦有成熟的商业软件工具和开源工具等。

1. 测试用例编写要素

一般测试用例应包括以下要素：

1) 名称和标识

每个测试用例应有唯一的名称和标识符。

2) 测试追踪

说明测试所依据内容的来源，如系统测试依据的是用户需求，配置项测试依据的是软件需求，集成测试和单元测试依据的是软件设计。

3) 用例说明

简要描述测试的对象、目的和所采用的测试方法。

4) 测试的初始化要求

应考虑下述初始化要求：①硬件配置(被测系统的硬件配置情况，包括硬件条件或电气状态)；②软件配置(被测系统的软件配置情况，包括测试的初始条件)；③测试配置(测试系统的配置情况，如用于测试的模拟系统和测试工具等的配置情况)；④参数设置(测试开始前的设置，如标志、第一断点、指针、控制参数和初始化数据等的设置)；⑤其他关于测试用例的特殊说明。

5) 测试的输入

在测试用例的执行中发送给被测对象的所有测试命令、数据和信号等。对于每个测试用例应提供如下内容：①每个测试输入的具体内容(如确定的数值、状态或信号等)及其性质(如有效值、无效值、边界值等)；②测试输入的来源(例如，测试程序产生、磁盘文件、通过网络接收、人工键盘输入等)，以及选择输入所使用的方法(例如，等价类划分、边界值分析、差错推测、因果图、功能图等方法)；③测试输入是真实的还是模拟的；④测试输入的时间顺序或事件顺序。

6) 期望的测试结果

说明测试用例执行中由被测软件产生的期望的测试结果，即经过验证，认为正确的结果。必要时，应提供中间的期望结果。期望的测试结果应该有具体内容，如确定的数值、状态或信号等，不应是不确切的概念或笼统的描述。

7) 评价测试结果的准则

判断测试用例执行中产生的中间结果和最后结果是否正确的准则。对于每个测试结果，应根据不同情况提供如下信息：①实际测试结果所需的精度；②对于实际测试结果与期望结果之间的差异，所允许的上限、下限；③时间的最大和最小间隔，或事件数目的最大和最小值；④实际测试结果不确定时，再测试的条件；⑤与产生测试结果有关的出错处理；⑥上面没有提及的其他准则。

8) 操作过程

实施测试用例的执行步骤。把测试的操作过程定义为一系列按照执行顺序排列的相对

独立的步骤，对于每个操作应提供：①每一步所需的测试操作动作、测试程序的输入、设备操作等；②每一步期望的测试结果；③每一步的评价准则；④程序终止伴随的动作或差错指示；⑤获取和分析实际测试结果的过程。

9) 前提和约束

在测试用例说明中施加的所有前提条件和约束条件，如果有特别限制、参数偏差或异常处理，应该标识出来，并说明它们对测试用例的影响。

10) 测试终止条件

说明测试正常终止和异常终止的条件。

11) 测试用例编写模板

在编写测试用例的过程中，需要参考和规范一些基本的测试用例编写标准。

12) ANSI/IEEE 829-1983 标准

在 ANSI/IEEE 829-1983 标准中列出了和测试设计相关的测试用例编写规范和模板。标准模板中的主要元素如下：

(1) 标识符(identification)：每个测试用例应该有一个唯一的标识符，它将成为所有和测试用例相关的文档/表格引用和参考的基本元素，这些文档/表格包括设计规格说明书、测试日志表、测试报告等。

(2) 测试项(test item)：测试用例应该准确地描述所需要测试的项及其特征，测试项应该比测试设计说明书中所列出的特性描述更加具体。例如，做 Windows 计算器应用程序的窗口设计，测试对象是整个应用程序的用户界面，这样测试项就应该是应用程序的界面特性要求，例如缩放测试、界面布局、菜单等。

(3) 测试环境(test environment)要求：用来表征执行测试用例需要的测试环境，一般来说，在整个测试模块里面应该包含整个测试环境的特殊要求，而单个测试用例的测试环境需要表征该测试用例所单独需要的特殊环境需求。

(4) 输入标准(input criteria)：用来执行测试用例的输入需求。这些输入可能包括数据、文件或操作(例如鼠标的左键单击、按键处理等)，必要的时候，相关的数据库、文件也必须被罗列。

(5) 输出标准(output criteria)：标识按照指定的环境和输入标准得到的期望输出结果。如果可能的话，尽量提供适当的系统规格说明书来证明期望的结果。

(6) 测试用例之间的关联：用来标识测试用例与其他测试(或其他测试用例)之间的依赖关系。例如，用例 A 需要基于用例 B 的测试结果正确的基础上才能进行，此时需要在用例 A 的测试用例中表明对用例 B 的依赖性，从而保证测试用例的严谨性。

13) 某公司的测试用例编写规范

(1) 测试用例命名规则：以功能模块和业务流程进行命名。

(2) 测试用例编号规则：对测试模块名称的第一个字母进行命名(大写)。若测试模块名称比较长，可进行简写。一般简拼不超过 5 个字母，例如：测试模块为用户管理，功能编号为 YHGL；测试模块为行政单位管理，功能编号为 DWGL；功能编号也可以直接以 001、002、003、……命名。

(3) 测试用例文档书写内容：①被测对象介绍及测试范围与目的；②测试环境与测试辅助工具的描述；③功能测试用例主要元素；④前置条件(可选)及操作，如系统权限配置或前后台配置描述(所有进行操作的前提条件)和测试的操作步骤描述；⑤功能点，即功能点描述；⑥输入数据，即前期数据准备；⑦预期结果，即描述输入数据后程序应该输出的结果；⑧测试结果，即描述本测试用例的实际测试情况，并判断实际测试结果与预期结果的差别；⑨bug 编号/bug 简要描述；⑩备注，即测试过程中遇到的问题等情况说明。

14) 测试用例编写实例

以常见的 Web 登录页面测试为例(当用户没有输入用户名和密码时，不立即弹出错误对话框，而是在页面上使用红色字体来提示；用户密码使用掩码*来标识；*代表必选字段，将出现在输入文本框的后面；当登录出现错误时，在页面的顶部会出现相应的错误提示，错误提示是高亮的红色字体)，下面给出某公司编写的测试用例，见表 8-2。

表 8-2 Web 登录页面的测试用例

字段名称	描述
标识符	1100
测试项	站点用户登录功能测试
测试环境要求	用户 pass/pass 为有效登录用户，用户 pass1/pass 为无效登录用户，pass’jean/password 为有效登录用户 浏览器的 Cookie 未被禁用
输入标准	输入正确的用户名和密码，单击“登录”按钮 输入错误的用户名和密码，单击“登录”按钮 不输入用户名和密码，单击“登录”按钮 输入正确的用户名但不输入密码，单击“登录”按钮 输入带特殊字符(/、’、和＃，如 pass’jean)的用户名和密码，单击“登录”按钮 三次输入无效的用户名和密码，尝试登录 第一次登录成功后，重新打开浏览器登录，输入上次成功登录的用户名的第一个字符
输出标准	数据库中存在的用户(pass/pass、pass’jean/password)能正确登录 错误的或无效的用户登录失败，页面的顶部出现红色字体：“错误：用户名或密码输入错误” 用户名为空时，页面顶部出现红色提示字体：“请输入用户名” 当密码为空且用户名不为空时，页面顶部出现红色字体提示：“请输入密码” 含特殊字符(‘、/、“和#)的用户名，如数据库中有该记录，将能正确登录；如数据库中无该用户记录，将不能登录，校验过程和普通的字符相同，不能出现空白页面或脚本错误 三次无效登录后，第四次尝试登录会出现提示信息“您已经三次尝试登录失败，请重新打开浏览器进行登录，此后的登录过程将被禁止” 自动完成功能将被禁止，查看浏览器的 Cookie 信息，将不会出现上次登录的用户名和密码信息，第一次使用一个新账户登录时，浏览器将不会提示“是否记住密码以便下次使用”对话框 所有的密码均以*掩码显示
测试用例间的关联	1101(有效密码测试)

2. 编写测试用例的注意事项

编写测试用例有很多注意事项，这里给出的是某公司编写测试用例的 4 个注意事项。

1) 功能检查

功能检查主要检查：①功能是否齐全，例如增加、删除、修改，查询条件是否合理，用户使用是否方便；②功能是否多余；③功能是否可以合并；④功能是否可以再细分；⑤软件流程与实际业务流程是否一致；⑥软件流程能否顺利完成；⑦各个操作之间的逻辑关系是否清晰；⑧各个流程数据传递是否正确；⑨模块功能是否与需求分析及概要设计相符；⑩批量增加、批量修改，增加、修改等录入比较频繁的界面或录入数据量较多的界面，是否支持全键盘或全鼠标操作，并且使用通用的键实现数据字段的有序切换。

2) 面向用户的考虑

面向用户的考虑主要涉及：①软件操作方便性和易用性，例如按键次数是否最少，并且不受开发实现技术限制，而是以用户使用方便性和应用软件约定和通常的快捷键来实现提出合理建议。另外，面对用户的操作是否简单易学；②智能化考虑；③提示信息是否模糊不清或有误导作用。错误信息是否有用户语言风格的出错后续处理建议提示；④要求用户进行的操作是否多余，能否由系统代替。系统升级后，用户能否不做任何操作自动进行所有升级的数据、环境准备等工作，包括删除缓存等动作；⑤能否记忆操作的初始环境，无须用户每次都进行初始化设置；⑥是否不经确认就对系统或数据进行重大修改；⑦能否及时反映或显示用户操作结果；⑧操作是否符合用户习惯，比如热键；⑨各种选项的可用及禁用是否及时合理；⑩某些相似的操作能否做成通用模块。

3) 数据处理

数据处理包括三方面的内容：数据输入、数据处理以及数据输出。

(1) 数据输入主要包括：边界值、大于边界值、小于边界值、最大个数、最大个数加 1、最小个数、最小个数减 1、空值或空表、极限值、0 值、负数、非法字符、日期和时间控制、跨年度数据、数据格式，以及数据之间的关联性、逻辑性，数据范围、格式限制是否合乎日常情理，如年龄不应为负数，身份证号码的位数必须为 15 或 18 位且与性别严格相关联，与生日可以有区别(考虑到阴历、阳历的问题)但不相同时给予提示，私人电话号码的长度且国内电话只能有数字及短横线标识区号等。

(2) 数据处理主要包括：处理速度、处理能力、数据处理正确率、计算方式及计算结果准确性(要考虑数字精度的取舍问题、汇总数据与分项数据累加的误差问题等)。

(3) 数据输出结果主要包括：正确率、输出格式、预期结果、实际结果(金额数字可能要验证小写和大写的一致性，大写可能要测试多种金额的大写，包括没有整数的情况、没有小数的情况、带整数和小数的情况……)。

4) 软件流程测试

软件流程测试主要考虑：反流程操作、反逻辑操作、重复操作、反业务流程操作以及违反流程的、打乱流程的或不按操作手册的乱操作。

8.4.3　测试用例的设计步骤

设计测试用例的时候，需要有清晰的测试思路，对要测试什么、按照什么顺序测试、覆盖哪些需求做到心中有数。测试用例编写者不仅要掌握软件测试的技术和流程，而且要对被测软件的设计、功能规格说明、用户使用场景以及程序/模块的结构都有比较透彻的理解。测试用例设计一般包括以下几个步骤：

1) 测试需求分析

从软件需求文档中找出待测试软件/模块的需求，通过自己的分析、理解，整理成为测试需求，清楚被测试对象具有哪些功能。

测试需求应该在软件需求基础上进行归纳、分类或细分，方便测试用例设计。测试用例中的测试集与测试需求的关系是多对一的，即一个或多个测试用例集对应一个测试需求。测试需求的特点是：包含软件需求，具有可测试性。

2) 业务流程分析

软件测试，不单纯是基于功能的黑盒测试，还需要对软件的内部处理逻辑进行测试。为了不遗漏测试点，需要清楚地了解软件产品的业务流程。建议在做复杂的测试用例设计前，先画出软件的业务流程。如果设计文档中已经有业务流程设计，可以从测试角度对现有流程进行补充。如果无法从设计中得到业务流程，测试工程师应通过阅读设计文档，与开发人员交流，最终画出业务流程图。业务流程图可以帮助理解软件的处理逻辑和数据流向，从而指导测试用例的设计。

从业务流程应得到以下信息：主流程是什么，条件备选流程是什么，数据流向是什么，以及关键的判断条件是什么。

3) 测试用例设计

完成了测试需求分析和软件流程分析后，开始着手设计测试用例。测试用例设计的类型包括功能测试、边界测试、异常测试、性能测试、压力测试等。在用例设计中，除了功能测试用例外，应尽量考虑边界、异常、性能的情况，以便发现更多的隐藏问题。

黑盒测试的测试用例设计方法有等价类划分、边界值划分、因果图分析和错误猜测，白盒测试的测试用例设计方法有语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、多重条件覆盖。在这里主要讨论黑盒测试。在设计测试用例的时候可以使用软件测试用例设计方法，结合前面的需求分析和软件流程分析进行设计：

(1) 功能测试：测试某个功能是否满足需求的定义，功能是否正确、完备。适合的技术：由业务需求和设计说明导出的功能测试、等价类划分。

(2) 边界测试：对某个功能的边界情况进行测试。适合的技术：边界值划分。

(3) 异常测试：对于某些功能来说，其边界情况无法简单地了解或某些操作不完全是正确的，但又是可能发生的，类似这样的情况需要书写相关的异常测试。适合的技术：由业务需求和设计说明导出的特殊业务流程、错误猜测法、边界值分析、内部边界值测试。

(4) 性能测试：检查系统是否满足在需求中所规定达到的性能，性能主要包括了解程序的内外部性能因素。内部性能因素包括测试环境的配置、系统资源使用状况；外部因素包括响应时间、吞吐量等。适合的技术：业务需求和设计说明导出的测试。

(5) 压力测试：压力测试又称强度测试，主要是检查系统运行环境在极限情况下软件运行的能力，比如给予相当大的负荷或网络流量给应用软件。

(6) 兼容测试：测试软件产品在不同平台、不同工具以及相同工具的不同版本下功能的兼容性。

4) 测试用例评审

测试用例设计完成后，为了确认测试过程和方法是否正确，以及是否有遗漏的测试点，需要进行测试用例的评审。

测试用例评审一般由测试主管安排，参加的人员包括：测试用例设计者、测试主管、项目经理、开发工程师、其他相关开发测试工程师。测试用例评审完毕后，测试工程师根据评审结果，对测试用例进行修改，并记录修改日志。

5) 测试用例更新完善

测试用例编写完毕之后需要不断完善，软件产品新增功能或更新需求后，测试用例必须配套修改更新；在测试过程中发现设计测试用例时考虑不周，需要对测试用例进行修改完善；对于在软件交付使用后客户反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成的情况，也需要对测试用例进行完善。一般小的修改完善可在原测试用例文档中修改，但文档要有更改记录。软件的版本升级更新，测试用例一般也应随之编制升级更新版本。测试用例是活的，在软件的生命周期中不断更新与完善。

8.4.4　测试用例分级

测试用例级别表明测试用例的重要程度。定义测试用例的级别，即定义测试用例在测试执行过程中的重要程度，包括优先执行。测试用例的重要性并不对应测试用例可能造成的后果，而是对应测试用例的基本程度，一个可能导致死机的测试用例未必是高级别的，因为其触发条件可能相当特别。

1. 测试用例的级别

在测试用例列表中有一项内容是用例级别，如表 8-3 所示。

表 8-3　测试用例列表

测试项目	测试子项目	用例编号	用例级别	输入值	预期输出	实测结果	备注
总数		——				——	——

测试用例一般分为 4 级。

1) 第 1 级：基本

(1) 该级别的测试用例涉及被测系统的基本功能，测试用例的数量应受到控制。

(2) 该级别的划分依据是测试用例执行失败会导致多项重要功能无法运行，如表单维

护中的增加功能、最平常的业务使用等。可以认为是发生概率较高且经常使用的一些功能。

(3) 该级别的测试用例在每一轮版本测试中都必须执行。

2) 第 2 级：重要

(1) 该级别的测试用例测试被测系统的重要功能，因此该级别的测试用例数量较多。

(2) 该级别的划分依据主要是针对一些功能交互相关、有各种应用场景、使用频率较高的正常功能测试用例。

(3) 在非回归的系统测试版本中基本上都需要进行验证，以保证系统所有的重要功能都是正常实现的。在测试过程中可以根据当前版本的具体情况决定是否在回归测试中全部执行或部分执行这些测试用例。

3) 第 3 级：一般

(1) 该级别的测试用例涉及系统的一般功能，因此该级别的测试用例数量也较多。

(2) 该级别的划分依据是针对使用频率低于第 2 级的测试用例。例如：数值或数组使用的便捷情况、特殊字符、字符串超长、与外部交互消息失败、消息超时、事务完整性测试、可靠性测试等。

(3) 在非回归的系统测试版本中不一定都进行验证，而且在系统测试的中后期并不一定需要每个版本都进行测试。

4) 第 4 级：特殊(如果没有，可以不使用该级别)

(1) 该级别的测试用例数量一般非常少。

(2) 该级别的划分依据是测试用例对应较特殊的预置条件和数据设置。虽然某些测试用例发现过较严重的错误，但是那些测试用例的触发条件非常特殊，仍然应该被置于第 4 级测试用例中。比如：界面规范化的测试也可归入第 4 级测试用例；在实际测试中，使用频率非常低、对用户可有可无的功能测试用例。

(3) 在版本测试中，某些正常原因(包括环境、人力、时间等)经测试经理同意可以不进行测试。

2. 测试用例的优先级

定义测试用例的级别是为了说明测试用例在测试执行时的优先程度以及用例的重要程度，而测试用例执行的优先程度首先取决于测试用例所测试的用户需求点的紧急程度、使用频率以及重要程度。测试用例的优先级定义首先要继承测试需求点的优先级别，所以首先应对整理的测试需求进行优先级定义，然后再对需求点对应的系列测试用例的优先级别进行定义。

在根据用户需求和需求分析文档提取测试需求时，必须知道所有需求中，哪些是用户急需使用的部分，哪些是用户使用频繁的部分，哪些是系统最不能出现错误的部分，等等，那么这些部分就是测试优先级最高的需求点。为此，在定义测试用例优先级时，应该考虑：①继承测试需求的优先级别；②测试用例在用户实际使用中的使用概率及频率；③测试用例导致被测软件出错的概率；④测试用例导致系统发生错误后对系统的危害性。

当然，测试用例的级别不是一旦定下来就不变的，根据实际测试过程中的实际情况，是可以变化的。例如：最初定义级别低的一个测试用例，由于在实际测试中，导致被测软

件出错的概率比较高，那么就应当在本轮测试结束前提高该测试用例的级别。再如：最初定义级别高的一个测试用例，由于在实际测试中导致被测软件出错的触发条件或操作组合或用户使用频率非常不易达到，或者由于用户需求变更，导致此需求点变为使用概率非常低的功能点，那么我们就应当在本轮测试结束前降低该测试用例的级别，等等。

8.4.5　软件测试用例设计的误区

测试的目的是尽可能发现程序中存在的缺陷，测试活动本身也可以被看作项目，也需要在给定的资源条件下尽可能达成目标。目前国内大部分的软件公司在测试方面配备的资源是不足的，因此我们必须在测试计划阶段明确测试的目标，一切围绕测试的目标进行；而在测试设计阶段，在围绕测试目标的基础上进行测试用例的设计。软件测试用例是为了有效发现软件缺陷而编写的包含测试目的、测试步骤、期望测试结果的特定集合。正确认识和设计软件测试用例可以提高软件测试的有效性，便于测试质量的度量，增强测试过程的可管理性。

在实际软件项目测试过程中，由于对软件测试用例的作用和设计方法的理解不同，测试人员(特别是刚刚从事软件测试的新人)对软件测试用例存在不少错误的认识，给实际软件测试带来了负面影响，下面我们就对这些认识误区进行列举和分析，避免在今后的测试用例设计过程中犯类似的错误：

1) 能发现到目前为止没有发现的缺陷的测试用例是好的测试用例

测试本身是一种 V&V 的活动，测试需要确保程序做了它应该做的事情，还要确保程序没有做它不该做的事情。因此，作为测试实施依据的测试用例，必须能够完成覆盖测试需求，而不应该针对单个测试用例去评判好坏。

2) 测试输入数据设计方法等同于测试用例设计方法

很多人认为软件测试用例的设计方法主要是：等价类划分、边界值分析、因果图、错误推测法、场景设计法等。这种表述是很片面的，这些方法只是软件功能测试用例设计中用于确定测试输入数据的方法，而不是测试用例设计的全部内容。

这种错误认识可能会使不少人认为测试用例设计就是如何确定测试的输入数据，从而掩盖了测试用例设计内容的丰富性和技术的复杂性。

无疑，对于软件功能测试和性能测试，确定测试的输入数据很重要，这决定了测试的有效性和测试效率。但是，在测试用例中输入数据的确定方法只是测试用例设计方法的一个子集，除了确定测试输入数据之外，测试用例的设计还包括如何根据测试需求、设计规格说明等文档确定测试用例的设计策略、设计测试用例的表示方法和组织管理形式等问题。

在设计测试用例时，需要综合考虑被测软件的功能、特性、组成元素、开发阶段(里程碑)、测试用例组织方法(是否采用测试用例的数据库管理)等内容。具体到设计每个测试用例而言，可以根据被测模块的最小目标，确定测试用例的测试目标；根据用户使用环境确定测试环境；根据被测软件的复杂程度和测试用例执行人员的技能确定测试用例的步骤；根据软件需求文档和设计规格说明确定期望的测试用例执行结果。

3) 强调测试用例设计越详细越好

在确定测试用例设计目标时，一些项目管理人员强调测试用例越详细越好。具体表现

在两个方面：一是尽可能设计足够多的设计用例，测试用例的数量越多越好；二是测试用例尽可能包括测试执行的详细步骤，达到任何一个人都可以根据测试用例执行测试的程度，追求测试用例越详细越好。

软件测试受到时间、人力和资金等资源的约束，而这种做法和观点最大的危害就是耗费了很多的测试用例设计时间和资源，可能等到测试用例设计、评审完毕后，留给实际执行测试的时间所剩无几。因为当前软件公司的项目团队在规划测试阶段时，分配给测试的时间和人力资源是有限的，而软件项目的成功要坚持质量、时间、成本的最佳平衡，没有足够多的测试执行时间，就无法发现更多的软件缺陷，测试质量更无从谈起。

编写测试用例的根本目的是有效地找出软件中可能存在的缺陷，为了达到这个目的，需要分析被测软件的特征，运用有效的测试用例设计方法，尽量使用较少的测试用例，同时满足合理的测试需求覆盖，从而达到少花时间多办事的效果。

测试用例中的测试步骤需要详细到什么程度，主要取决于测试用例的最终用户(即执行这些测试用例的人员)，以及测试用例执行人员的技能和产品熟悉程度。在测试计划阶段，一般给予测试设计 30%~40%的时间，测试设计工程师能够根据项目的需要自行确定测试用例的详细程度，在测试用例的评审阶段由参与评审的相关人对其把关。总之，文档的作用主要用于沟通，只要能达到沟通的目的就可以了。

4) 追求测试用例设计一步到位

现在软件公司都意识到了测试用例设计的重要性了，但是一些人认为设计测试用例是一次性投入，测试用例设计一次就万事大吉了，片面追求测试设计的一步到位。

这种认识造成的危害性是使设计出的测试用例缺乏实用性，或者误导测试用例执行人员，误报很多不是软件缺陷的 bug，这样的测试用例在测试执行过程中形同虚设，难免沦为垃圾文档。

“唯一不变的是变化”。任何软件项目的开发过程都处于不断变化过程中，用户可能对软件的功能提出新需求，设计规格说明也相应地更新，软件代码不断细化。设计软件测试用例与软件开发设计并行进行，必须根据软件设计的变化，对软件测试用例进行内容的调整以及数量的增减，增加一些针对软件新增功能的测试用例，删除一些不再适用的测试用例，修改那些模块代码更新的测试用例。

软件测试用例设计只是测试用例管理的一个过程，除此之外，还要对其进行评审、更新、维护，以便提高测试用例的“新鲜度”，保证可用性。因此，软件测试用例也要坚持与时俱进的原则。

5) 测试用例不应该包含实际的数据

测试用例是一组输入、执行条件、预期结果，因此测试用例毫无疑问地应该包括清晰的输入数据和预期输出，没有测试数据的用例最多只具有指导意义，不具有可执行性。当然，测试用例中包含输入数据会带来维护问题，以及与测试环境同步之类的变化问题。

6) 测试用例中不需要明显的验证手段

预期输出仅描述为程序的可见行为，其实，预期结果的含义并不只是程序的可见行为。例如，对于一个订货系统，输入订货数据，单击“确定”按钮后，系统提示“订货成功”，这是不是一个完整的测试用例呢？是不是系统输出的“订货成功”就应该作为我们唯一的

验证手段呢？显然不是。订货是否成功还需要查看相应的数据记录是否更新，因此，在这样的测试用例中，还应该包含对测试结果的显式的验证手段：在数据库中执行查询语句进行查询，看看查询结果是否与预期的一致。

7) 让测试新人设计测试用例

刚参加测试工作的测试新手经常思考和询问的一个问题是：“怎么才能设计好测试用例？”因为他(她)们以前从来没有设计过测试用例，所以在面对大型的被测试软件时感到“老虎吃天，无从下口”。

让测试新手设计测试用例是一种高风险的测试组织方式，带来的不利后果是设计出的测试用例对软件功能和特性的测试覆盖性不高，编写效率低，审查和修改时间长，可重用性差。而事实上，软件测试用例设计是软件测试的中高级技能，不是每个人(尤其是测试新人)都可以编写的，测试用例编写者不仅要掌握软件测试的技术和流程，而且要对被测软件的设计、功能规格说明、用户使用场景以及程序/模块的结构都有比较透彻的理解。

因此，在实际测试过程中，通常安排经验丰富的测试人员进行测试用例的设计，测试新人可以从执行测试用例开始，随着项目进度的不断进展，测试人员的测试技术和对被测软件的不断熟悉，可以积累测试用例的设计经验，编写测试用例。

8.4.6　软件测试用例设计举例

对于高校学生选课管理系统的测试用例设计，可以按照系统各功能模块进行设计。本系统主要包括用户登录、专业管理、课程管理、统计信息、修改密码五大功能模块，测试用例设计可以相应分为五大部分，每部分测试用例按照功能需求逐一展开，要求覆盖所有功能点，如表 8-4 所示。

表 8-4　测试用例设计举例

1. 用户登录功能		
测试步骤		
操作说明	预期结果	实测结果
打开登录界面，在用户名文本框输入 mr，在密码文本框中输入 mrsoft，单击“登录”按钮	页面跳转到高校学生选课管理系统主界面	
打开登录界面，在用户名文本框中输入 mr，保持密码文本框为空，单击“登录”按钮	页面不跳转，用户名文本框上方显示红字“用户登录失败，用户名或密码不正确！”	
打开登录界面，在用户名文本框中输入 mr，在密码文本框中输入 123456，单击“登录”按钮	页面不跳转，用户名文本框上方显示红字“用户登录失败，用户名或密码不正确！”	
2. 专业管理功能		
测试步骤		
操作说明	预期结果	实测结果
打开高校学生选课管理系统主界面，单击“专业管理”按钮	页面跳转到“专业管理”界面	

(续表)

<table>
<tr><td colspan="2">打开“专业管理”界面，单击“增加新专业”按钮</td><td>页面跳转到“添加新专业”界面</td><td></td></tr>
<tr><td colspan="2">打开“添加新专业”界面，在“入学年份”下拉栏中选择 2005 年，在“专业名称”文本框中输入“软件工程”，在“学制”下拉栏中选择 4 年，单击“确定”按钮</td><td>页面跳转到“专业管理”界面，新专业已经出现在专业清单中</td><td></td></tr>
<tr><td colspan="2">打开“添加新专业”界面，在未结业的专业的操作栏单击“设置为已结业”</td><td>在此界面，此专业的操作栏显示为“已结业”</td><td></td></tr>
<tr><td colspan="4">3. 课程管理功能</td></tr>
<tr><td colspan="4">测试步骤</td></tr>
<tr><td colspan="2">操作说明</td><td>预期结果</td><td>实测结果</td></tr>
<tr><td colspan="2">打开高校学生选课管理系统主界面，单击“课程管理”按钮</td><td>页面跳转到“课程管理”界面</td><td></td></tr>
<tr><td colspan="2">打开“课程管理”界面，在“请输入搜索条件”框内，在“选择专业”框中选择某专业，在“授课教师姓名”框内输入此专业的授课教师姓名，在“课程名称”框中输入课程名称，单击“搜索”按钮</td><td>页面跳转到符合条件的课程清单</td><td></td></tr>
<tr><td colspan="2">打开“课程管理”界面，在“课程名称”栏单击“大学英语 1”</td><td>页面跳转到“大学英语 1”的课程详细信息：专业信息，课程信息，教师授课信息</td><td></td></tr>
<tr><td colspan="2">打开“课程管理”界面，单击“增加新课程”按钮</td><td>页面跳转到“添加新课程”界面</td><td></td></tr>
<tr><td rowspan="2">打开“添加新课程”界面，根据所要求添加的新课程信息进行添加：选择专业、课程名称、上课时间、上课地点、课程学分、课程介绍、授课教师和教师介绍</td><td>单击“确定”按钮</td><td>页面跳转到该专业新添加的课程清单</td><td rowspan="2"></td></tr>
<tr><td>单击“重置”按钮</td><td>当前页面输入的信息全部清零</td></tr>
<tr><td colspan="2">打开“课程管理”界面，选择“学生可选”栏中某个可选的课程，单击此课程的课程名称</td><td>页面跳转到“课程详细信息”界面</td><td></td></tr>
<tr><td colspan="2">打开“课程详细信息”界面，在“课程是否可选”行，单击“设置为不可选”</td><td>页面上的“课程是否可选”行显示为“否”</td><td></td></tr>
<tr><td colspan="4">4. 统计管理功能</td></tr>
<tr><td colspan="4">测试步骤</td></tr>
<tr><td colspan="2">操作说明</td><td>预期结果</td><td>实测结果</td></tr>
<tr><td colspan="2">打开高校学生选课管理系统主界面，单击“统计管理”按钮</td><td>页面跳转到“统计信息”界面</td><td></td></tr>
<tr><td colspan="2">打开“统计信息”界面，在“输入搜索条件”框内，在“选择专业”框中选择某专业，在“授课教师姓名”框内输入此专业的授课教师姓名，在“课程名称”框中输入课程名称，单击“搜索”按钮</td><td>页面跳转到符合条件的课程清单</td><td></td></tr>
</table>

(续表)

<table>
<tr><td colspan="2">在符合搜索条件的课程清单界面中，选择“计算机二级”课程，单击“查看</td><td>页面跳转到“计算机二级课程听课人员名单”界面，显示出当前课程所有上课学生的清单</td><td></td></tr>
<tr><td colspan="2">在“课程听课人员名单”界面，单击“导出 PDF 文档”(Excel 文档)</td><td>当前页面跳出“计算机二级.pdf(excel)文档保存”窗口</td><td></td></tr>
<tr><td colspan="4">5. 修改密码功能</td></tr>
<tr><td colspan="4">测试步骤</td></tr>
<tr><th colspan="2">操作说明</th><th>预期结果</th><th>实测结果</th></tr>
<tr><td colspan="2">打开高校学生选课管理系统主界面，单击“修改密码”按钮</td><td>页面跳转到“修改密码”界面</td><td></td></tr>
<tr><td rowspan="2">打开“修改密码”界面，根据所要添加的账号信息进行输入：登录账号 mr，旧密码 mrsoft，新密码 123456，确认密码 123456，验证邮箱 xyz@163.com</td><td>单击“确认”按钮</td><td>密码修改成功</td><td rowspan="2"></td></tr>
<tr><td>单击“重置”按钮</td><td>当前页面输入的信息全部清零</td></tr>
<tr><td colspan="2">打开“修改密码”界面，登录账号 mr，旧密码 mrsoft，新密码 123456，确认密码 123，验证邮箱 xyz@163.com</td><td>页面不跳转，用户名文本框上方显示红字“输入的信息不正确，不能修改密码”</td><td></td></tr>
<tr><td colspan="2">打开“修改密码”界面，登录账号 mr，旧密码 mrsoft，新密码 123456，确认密码为空，验证邮箱 xyz@163.com</td><td>页面不跳转，用户名文本框上方显示红字“输入的信息不正确，不能修改密码”</td><td></td></tr>
<tr><td colspan="2">打开主界面，单击“退出系统”按钮</td><td>页面跳转到登录界面</td><td></td></tr>
</table>

习题和思考题

1. 什么是动态测试？动态测试包括哪些内容？
2. 什么是白盒测试？白盒测试采用哪些方法？我们如何进行白盒测试？
3. 什么是逻辑覆盖、路径测试？它们包含哪些内容？
4. 什么是数据流测试？我们如何进行数据流测试？
5. 什么是信息流测试？信息流测试包含哪些内容？
6. 什么是黑盒测试？黑盒测试一般采用哪些方法？我们如何进行黑盒测试？
7. 什么是等价类划分法、边界值分析法？我们如何应用这些方法进行黑盒测试？
8. 什么是因果图分析法？我们如何用因果图生成测试用例？
9. 测试用例设计的原则和要素是什么？我们如何进行测试用例的设计？
10. 简述测试用例分级及测试用例优先级的概念。

第9章　软件单元测试

软件单元是指软件设计说明中一个可独立测试的元素，是程序中一个逻辑上独立的部分，它不能再分解为其他软件成分，如软件源代码中单个的函数、源文件或类。

单元测试(模块测试)是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，单元测试是用于判断某个特定条件(或场景)下某个特定函数的行为，是对单个软件单元或一组相关的软件单元所进行的测试，是代码级的测试。

按照软件生命周期对软件测试所进行的级别划分，单元测试是最初始级别的测试，然后是集成测试(组装测试)、确认测试(配置项测试)和系统测试。图 9-1 表示了测试步骤间的关系。

开始是单元测试，集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确实现了程序规定的功能。然后把已经测试过的程序模块组装起来，进行集成测试，主要对与设计相关的软件体系结构的构造进行测试。在这里，将一个一个经过单元测试并确保无误的程序模块组装成软件系统，对其正确性和程序结构方面进行检查。确认测试则是检查已经组装好的软件系统是否满足需求规格说明中明确说明的各种需求，以及软件配置是否安全、正确。最后是系统测试，把经过确认测试的软件在实际环境中运行，并与其他系统组合在一起进行测试。

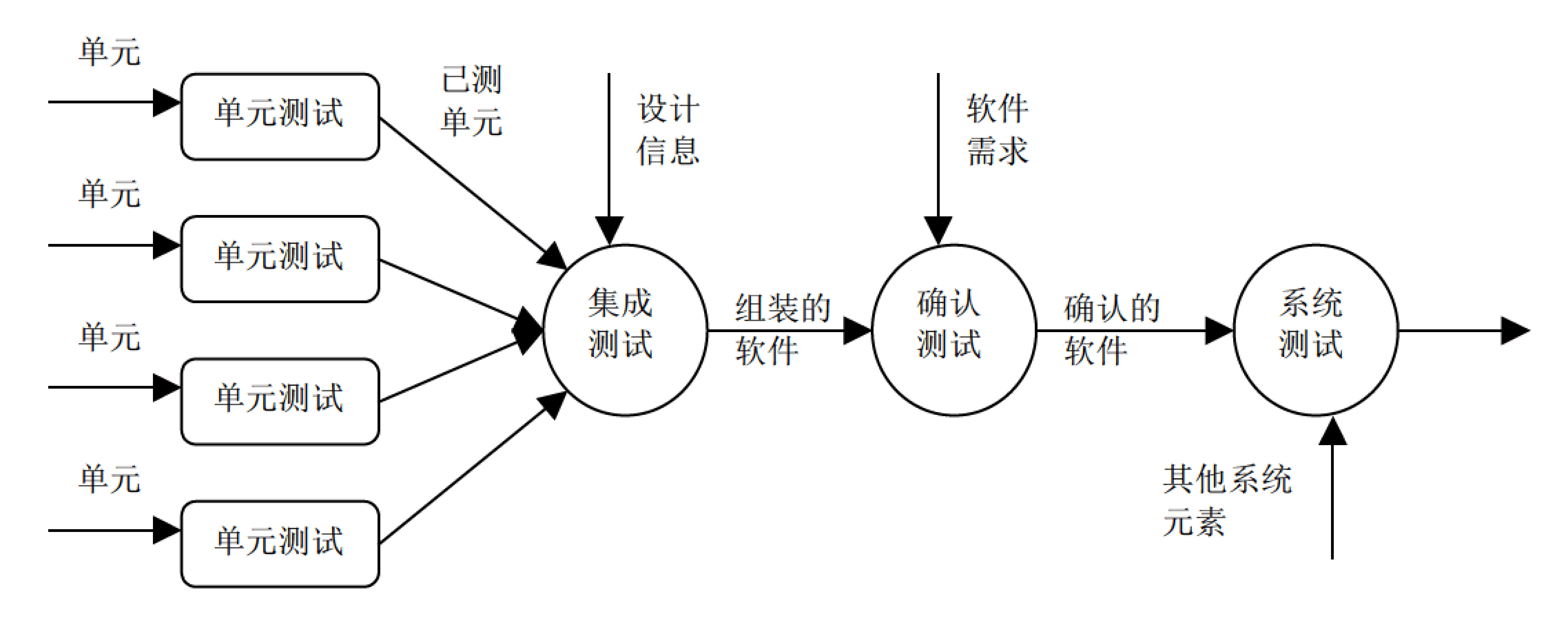


图 9-1　软件测试步骤

除对应与软件开发过程逐步进行验证和确认之外，软件测试分步骤进行的另一个意义在于，不同阶段的测试对应不同层次的软件阶段产品，可以针对测试对象不同的特点采用不同的技术，发现不同特征的错误，使软件在多层次的测试中不断提高质量。

单元测试又称模块测试，是针对软件设计的最小单位——程序模块或功能模块，进行正确性检验的测试工作。其目的在于检验程序各模块中是否存在各种差错，是否能正确地实现其功能，满足其性能和接口要求。单元测试如果认真实施效果会非常好，但是实施阻

力比较大(主要是人员和管理因素)，因此很多开发团队一般只在关键的程序单元中实施。单元测试有比较系统的理论和方法，但也依赖于系统的特殊性和开发人员的经验。单元测试有大量的辅助工具，开发人员也经常自己开发测试代码和测试工具。

单元测试验证程序和详细设计说明的一致性，需要从程序的内部结构出发设计测试用例，多个模块之间可以平行独立地进行单元测试。

程序模块是由汇编程序、编译程序、装入程序或翻译程序作为整体来处理的一级独立的、可识别的程序指令，是大型程序指令的组成部分；功能模块是指实现了一个完整功能的程序(单元)，一个完整的程序单元具备输入、加工和输出三个环节，每个程序单元都应该有正规的规格说明，要对其输入、加工和输出的关系做出明确描述。

9.1　单元测试概述

9.1.1　单元测试的意义

1. 对单元测试的误解

对于单元测试，人们往往存在很多的误解。

1) 浪费的时间太多

一旦编码完成，缺乏软件工程实践经验的开发人员就会迫不及待地进行软件集成工作，这样就能看到实际系统开始启动工作，在这种开发步骤中，意义上的进步被表面上的进步所取代。现实中发现编码阶段引入的缺陷远远多于其他阶段，系统测试发现的缺陷大多数是编码缺陷。这样，系统能进行正常工作的可能性很小，更多的情况是充满了各式各样的 bug。这些 bug 包含在独立的单元里，其本身也许是琐碎、微不足道的，但在软件集成为系统时会增加额外的工期和费用。其实进行完整的单元测试和编写代码所花费的精力大致上是相同的，一旦完成单元测试，确保手头拥有稳定可靠部件的情况下，再进行高效的软件集成才是真正意义上的进步。

程序的可靠性对软件产品的质量有很大的影响，在大型软件公司里，每写一行程序，都可能要测试很多遍。由此可见大型软件公司对测试的重视程度。

2) 软件开发人员不应参与单元测试

目前很多开发团队的开发人员承担着包括设计、编码及测试多个角色(如参与或部分参与高层设计，承担低层设计或程序实现，担当低层测试人员等)。他们强调开发人员做测试时间紧、效果不好，而且也不知道怎么做，等等。但这些都不是理由，因为单元测试常常和编码同步进行，每完成一个模块就应进行单元测试。在对每个模块进行单元测试时，不能忽略和其他模块的关系，为模拟这一关系，需要辅助模块，因此若单独的测试人员进行单元测试，往往工作量大，周期长，耗费巨大，其结果事倍功半。单元测试由程序员自己来完成，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试，以保证它们实现了设计的功能(其实在很多情

况下，开发者也应进行集成测试)。另外，对于程序员来说，如果养成了对自己所写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且还能提高编程水平。

3) 它们仅仅证明这些代码做了什么

这是那些没有首先为每个单元编写详细的规格说明而直接跳到编码阶段的开发人员提出的普遍抱怨，编码完成以后并且在面临代码测试任务的时候，他们就阅读这些代码并找出它们实际上做了什么，把他们的测试工作基于已经写好的代码的基础上。当然，它们无法证明任何事情。所有的这些测试工作能够表明的事情就是编译通过的。是的，他们也许能够抓住(希望能够抓住)罕见的编译错误，但是他们能够做的仅仅是这些。

如果他们首先写好详细的规格说明，测试能够以规格说明为基础。代码就能够针对它们的规格说明，而不是针对自身进行测试。这样的测试仍然能够抓住编译错误，同时也能找到更多的编码错误，甚至是一些规格说明中的错误。好的规格说明可以使测试的质量更高，所以最后的结论是：高质量的测试需要高质量的规格说明。

4) 我是很棒的程序员，不需要进行单元测试

传统的开发观念是：开发人员的任务是完成编程，让系统正确运行起来。如果程序有错，就进行调试，程序通过调试任务就完成了。而且我是编程高手，自信自己的程序不会出错。

如果我们真正擅长编程并且有绝招，就应当不会有错误，但这只是一个神话。要记住的是：开发人员的任务是完成程序，直到交付和维护；另外，人的失误是不可避免的，无论多小心，都会有错误。因此，程序必须经过各种各样的测试，单元测试只是其中一种。

5) 不管怎样，集成测试或系统测试将会抓住所有的 bug

集成测试的目标是把通过单元测试的模块拿来，构造一个在设计中所描述的程序结构，通过测试发现和接口有关的问题。我们在测试工作开展的过程中，发现并提交进行合格性测试的软件，在测试过程中有很多 bug，有些严重问题，甚至导致死机，以至于不能再测试其他功能，进行错误修改，回归测试时又发现其他新的问题，使得测试工作很难开展下去。

如果把单元测试的任务堆积到系统测试阶段，致使大量的故障堆积在项目中后期(项目后10%的工作，占用了项目90%的时间)，造成故障难以定位或飘忽不定，开发及测试人员疲于奔命，费用成倍上升。

6) 单元测试效率不高

在实际工作中，开发人员不想进行单元测试，认为没有必要且效率不高，其实错误发生和被发现之间的时间与发现和改正错误的成本呈指数关系，频繁的单元测试能使开发人员排错的范围缩得很小，大大节约排错所需的时间，同时尽可能早地发现和消灭错误也将减少由于错误而引起的连锁反应。

在某一功能点上进行准备测试、执行测试和修改缺陷，单元测试的效率大约是集成测试的两倍、系统测试的三倍。

通过对这些误解的分析，我们对单元测试有了进一步的了解，其实作为软件系统的最小组成单位，单元测试具有以下属性：①它是由程序员完成的；②它有详细的设计说明，

包括输入定义、输出定义和加工说明；③它是可识别、看得见的程序组成部分，并容易被组合成程序；④能被单独地汇编和测试；⑤它的规模比较小，逻辑比较简单。

2. 单元测试的好处

1) 单元测试将注意力集中在程序的基本组成部分

首先保证每个单元测试通过，才能使下一步工作——把单元组装成部件并测试其正确性有基础。单元是整个软件的构成基础，像硬件系统中的零部件一样，只有保证零部件的质量，这台设备的质量才有基础，单元的质量也是整个软件质量的基础。因此，单元测试的效果会直接影响软件的后期测试，最终在很大程度上影响产品的质量。

2) 单元测试可以平行开展

这样可以使多人同时测试多个单元，提高了测试效率。

3) 单元规模较小，复杂性较低

因而发现错误后容易隔离和定位，有利于调试工作。另外，也使单元测试可以使用包括白盒测试在内的许多测试技术，能够进行比较充分细致的测试，确保整个程序测试满足一些基本覆盖要求。

4) 单元测试的测试效果是最显而易见的

做好单元测试，不仅后期的系统集成联调或集成测试和系统测试会很顺利，节约很多时间；而且在单元测试过程中能发现一些很深层次的问题，同时还会发现一些很容易发现而在集成测试和系统测试阶段很难发现的问题；更重要的是单元测试不仅仅证明这些代码做了什么，而且演示代码是如何做的，以及是否做了它该做的事情而没有做它不该做的事情。

5) 单元测试的好与坏直接关系到测试成本，也会直接影响到产品质量

单元测试中易发现的问题如果拖到后期测试才发现，其修复成本将成倍上升；另外，可能就是由于代码中的某个小错误而导致整个产品的质量降低一个指标，或者导致更严重的后果。

单元测试要求尽早、可重复地、尽可能采用自动化手段进行。事实上，单元测试是一种验证行为：测试和验证程序中每一项功能的正确性，为以后的开发提供支持。单元测试是一种设计行为：编写单元测试将使我们从调用者角度观察和思考，特别是要先考虑测试，这样就可把程序设计成易于调用和可测试的，并努力降低软件中的耦合，还可以使编码人员在编码时产生预测试，将程序的缺陷降到最少。单元测试是一种编写文档的行为：是展示函数或类如何使用的最佳文档。单元测试具有回归性：自动化的单元测试有助于回归测试的开展。

9.1.2　单元测试的内容

单元测试由一组独立的测试构成，每个测试针对软件中一个单独的程序单元。单元测试并非检查程序单元之间是否能够合作良好，而是检查单个程序单元行为是否正确。在进行单元测试时，测试人员根据详细设计说明书和源程序清单，了解到模块的I/O条件和逻辑结构，主要采用白盒测试的测试用例，辅之以黑盒测试的测试用例，使之对任何合理和不合理的输入都要能鉴别和响应。这就要求对程序所有的局部和全局的数据结构、外部接

口和程序代码的关键部分进行桌面检查和代码审查。

对被测模块或单元进行单元测试主要有 5 个方面的内容，如图 9-2 所示。

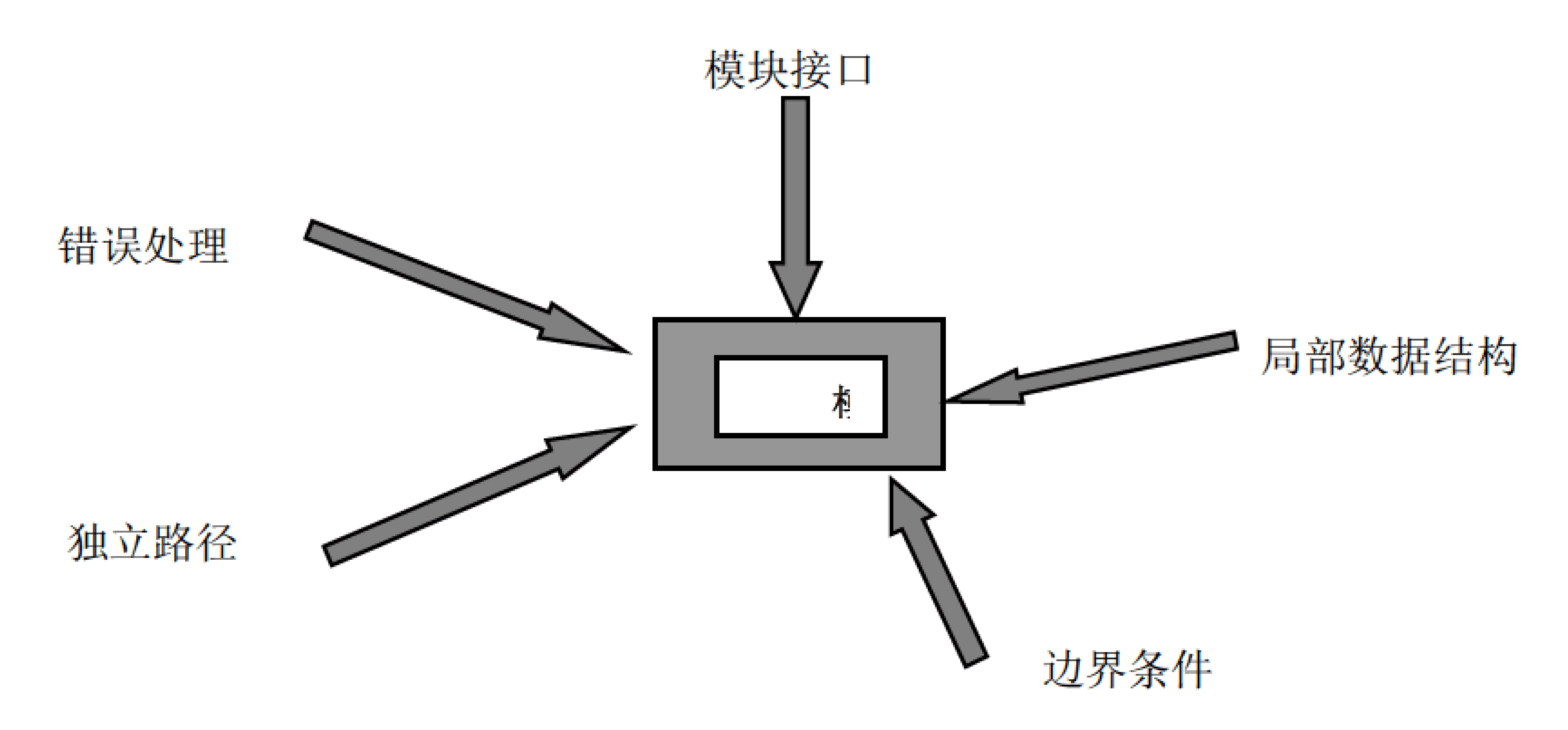


图 9-2　测试的内容

1. 单元测试 5 个方面的内容

1) 模块接口测试

在单元测试开始时，应该对通过所有被测模块的数据流进行测试。如果数据不能正常地进入及输出，那么其他的全部测试都说明不了问题。Myers 在他关于软件测试的书中为接口测试提出了一个检查列表：

(1) 模块输入参数的数目是否与模块形参数目相同？

(2) 模块各输入的参数属性与对应的形参属性是否一致？

(3) 模块各输入的参数类型与对应的形参类型是否一致？

(4) 传到被调用模块的实际参数的数目是否与被调用模块形参的数目相同？

(5) 传到被调用模块的实际参数的属性是否与被调用模块形参的属性相同？

(6) 传到被调用模块的实际参数的类型是否与被调用模块形参的类型相同？

(7) 引用内部函数时，实参的次序和数目是否正确？

(8) 是否引用了与当前入口无关的参数？

(9) 用于输入的变量有没有改变？

(10) 在经过不同模块时，全局变量的定义是否一致？

(11) 限制条件是否以形参的形式传递？

(12) 使用外部资源时，是否检查可用性并及时释放资源，如内存、文件、硬盘、端口等。

当模块通过外部设备进行输入/输出操作时，必须扩展接口测试，附加如下测试项目：

(1) 文件的属性是否正确？

(2) Open 与 Close 语句是否正确？

(3) 规定的格式是否与 I/O 语句相符？

(4) 缓冲区的大小与记录的大小是否相配合？

(5) 在使用文件前，文件是否打开？

(6) 文件结束的条件是否安排好了？

(7) I/O 错误是否检查并做了处理？

(8) 在输出信息中是否有文字错误？

2) 局部数据结构测试

模块的局部数据结构是最常见的错误来源，应设计测试用例以检查以下各种错误：

(1) 不正确或不一致的数据类型说明。

(2) 使用尚未赋值或尚未初始化的变量。

(3) 错误的初始值或默认值。

(4) 变量名拼写错或书写错——使用了外部变量或函数。

(5) 不一致的数据类型。

(6) 全局数据对模块的影响。

(7) 数组越界。

(8) 非法指针。

3) 独立路径测试

检查由于计算错误、判定错误、控制流错误导致的程序错误。由于在测试时不可能做到穷举测试，因此在单元测试中需要根据白盒测试和黑盒测试用例设计方法设计测试用例，对模块中重要的执行路径进行测试。重要的执行路径指那些处在完成单元功能的算法、控制、数据处理等重要位置的执行路径，也指由于控制较复杂而易错的路径，有选择地对执行路径进行测试是一项重要的任务。应当设计测试用例，查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误，对基本执行路径和循环进行测试可发现大量的路径错误。

在独立路径测试中，要检查的错误有：死代码，错误的计算优先级，算法错误，混用不同类的操作，初始化不正确，精度错误——比较运算错误、赋值错误，表达式的不正确符号(-->、>=、=、==、!=)，循环变量的使用错误——错误赋值以及其他错误等。

比较操作和控制流向紧密相关，测试用例设计需要注意发现比较操作的错误：

(1) 不同数据类型的比较。

(2) 不正确的逻辑运算符或优先次序。

(3) 因浮点运算精度问题而造成的两值比较不等。

(4) 关系表达式中不正确的变量和比较符。

(5) 差 1 错，即不正常的或不存在的循环中的条件。

(6) 当遇到发散的循环时无法跳出循环。

(7) 当遇到发散的迭代时不能终止循环。

(8) 错误地修改循环变量。

4) 错误处理测试

错误处理路径是可能引发错误处理的路径及进行错误处理的路径，错误出现时错误处理程序重新安排执行路线，或通知用户处理，或干脆停止执行使程序进入一种安全等待状态。测试人员应意识到，每一行程序代码都可能执行到，不能自己认为错误发生的概率很小而不去进行测试。一般软件错误处理测试应考虑下面几种可能的错误：

(1) 出错的描述是否难以理解，是否能够对错误定位。

(2) 显示的错误与实际的错误是否相符。

(3) 对错误条件的处理正确与否。

(4) 在对错误进行处理之前，错误条件是否已经引起系统的干预等。

在进行错误处理测试时，我们要检查如下内容：

(1) 在资源使用前后或其他模块使用前后，程序是否进行错误出现检查。

(2) 出现错误后，是否可以进行错误处理，如引发错误、通知用户、进行记录。

(3) 在系统干预前，错误处理是否有效，报告和记录的错误是否真实详细。

5) 边界条件测试

边界条件测试是单元测试中最后的任务。软件常常在边界上有错误，例如，在一个程序段中有一个 n 次循环，在到达第 n 次循环时就可能会出错；或者在一个有 n 个元素的数组中，第 n 个元素是很容易出错的。因此，要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。

此外，如果对模块性能有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响运行时间的因素。下面是边界条件测试的具体需要检查的内容：

(1) 普通合法数据是否正确处理。

(2) 普通非法数据是否正确处理。

(3) 边界内最接近边界的(合法)数据是否正确处理。

(4) 边界外最接近边界的(非法)数据是否正确处理等。

(5) n 次循环的第 0 次、第 1 次、第 n 次循环是否有错误。

(6) 运算或判断中取最大值/最小值时是否有错误。

(7) 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误。

2. 单元测试的要求

为了使单元测试能充分细致地展开，应在实施单元测试时遵守下述要求：

1) 语句覆盖达到 100%

语句覆盖指被测单元中每条可执行语句都被一个测试用例所覆盖。语句覆盖是强度最低的覆盖要求，要考虑语句覆盖的意义，只要想象一下用一段从没执行过的程序控制庞大的飞行器升上天空，然后设法使它精确入轨，这种轻率简直就是荒唐。实际测试中，不一定能做到每条语句都执行到。第一，存在死码，即由于程序设计错误，在任何情况下都不可能执行的代码。第二，不是死码，但是由于要求的测试输入及条件非常难达到或单元测试的条件所限，使得代码没有得到运行。因此，在可执行语句未得到执行时，要深入程序做详细分析。如果属于以上两种情况，则可以认为完成了覆盖；但是对于后者，如果可能，一定要尽量测试到；如果以上两者都不是，则是因为测试用例设计不充分，需要再次设计测试用例。

2) 分支覆盖达到 100%

分支覆盖指分支语句取真值和取假值各一次，分支语句是程序控制流的重要处理语句，在不同流向上测试可以验证这些控制流向的正确性。分支覆盖使这些分支产生的输出都得到验证，提高了测试的充分性。

3) 错误处理路径达到 100%

覆盖所有的错误处理路径。

4) 单元的软件特性覆盖

软件的特性包括功能、性能、属性、设计约束、状态数目、分支的行数等。

5) 各种数据特性覆盖

对使用额定数据值、奇异数据值和边界值的计算进行检验，用假想的数据类型和数据值运行，测试排斥不规则输入的能力。

单元测试通常是由编写程序的人自己完成的，但是项目负责人应当关心单元测试的结果。所有的测试用例和测试结果都是模块开发的重要资料，需要妥善保存。

9.2　单元测试方法和步骤

在软件开发过程中，代码编写和单元测试同属实现阶段，编码完成并编译通过后才开始进行单元测试。

进行动态的单元测试前，需要先对程序进行静态分析和代码审查。这是因为：

(1) 使用动态测试技术要准备测试用例，进行结果记录和分析，工作量大，发现错误太多会降低动态测试效率。

(2) 目前的动态测试技术局限性比较大，有相当类型的错误靠动态测试是难以发现的。因此，先使用静态分析和代码审查技术，能充分地发挥人的判断和思维优势，检查出对于机器而言很难发现的错误。典型的包括代码和设计规格的一致性、代码逻辑表达式的正确性。这些检查在动态测试阶段将会是非常烦琐而又非常困难的。

(3) 有些错误在动态测试时无法检查到。

(4) 使用代码审查技术，一旦发现错误，就知道错误的性质和位置，调试代价较低。

(5) 使用静态分析方法一次就能揭示一批错误，并且随后就可以立即纠正错误。

由于单元测试针对程序单元，而程序单元并不是独立可运行的程序，因此在考虑被测模块时，同时要考虑到它和外界其他模块的联系。

9.2.1　单元测试方法

在考虑被测单元和外界其他模块的联系时，可用一些辅助模块去模拟与被测模块关联的其他模块。这些模块分为两种：

(1) 驱动模块。相当于所测模块的主程序。它接收测试数据，把这些测试数据传送给被测模块，最后输出实测结果。

(2) 桩模块。由被测模块调用，用以代替由被测单元调用的模块的功能，返回适当的数据或进行适当的操作，使被测单元能继续运行下去，同时还要进行一定的数据处理，如打印入口和返回等，以便检验被测模块与其下级模块的接口。

驱动模块和桩模块为程序单元的执行构成了一个完整的环境，如图 9-3 所示。驱动模块用以模拟被测单元的上层模块，测试执行时由驱动模块调用被测单元使其运行，桩模块

模拟被测单元执行过程中所调用的模块,测试执行时桩模块使被测单元能完整闭合地运行。

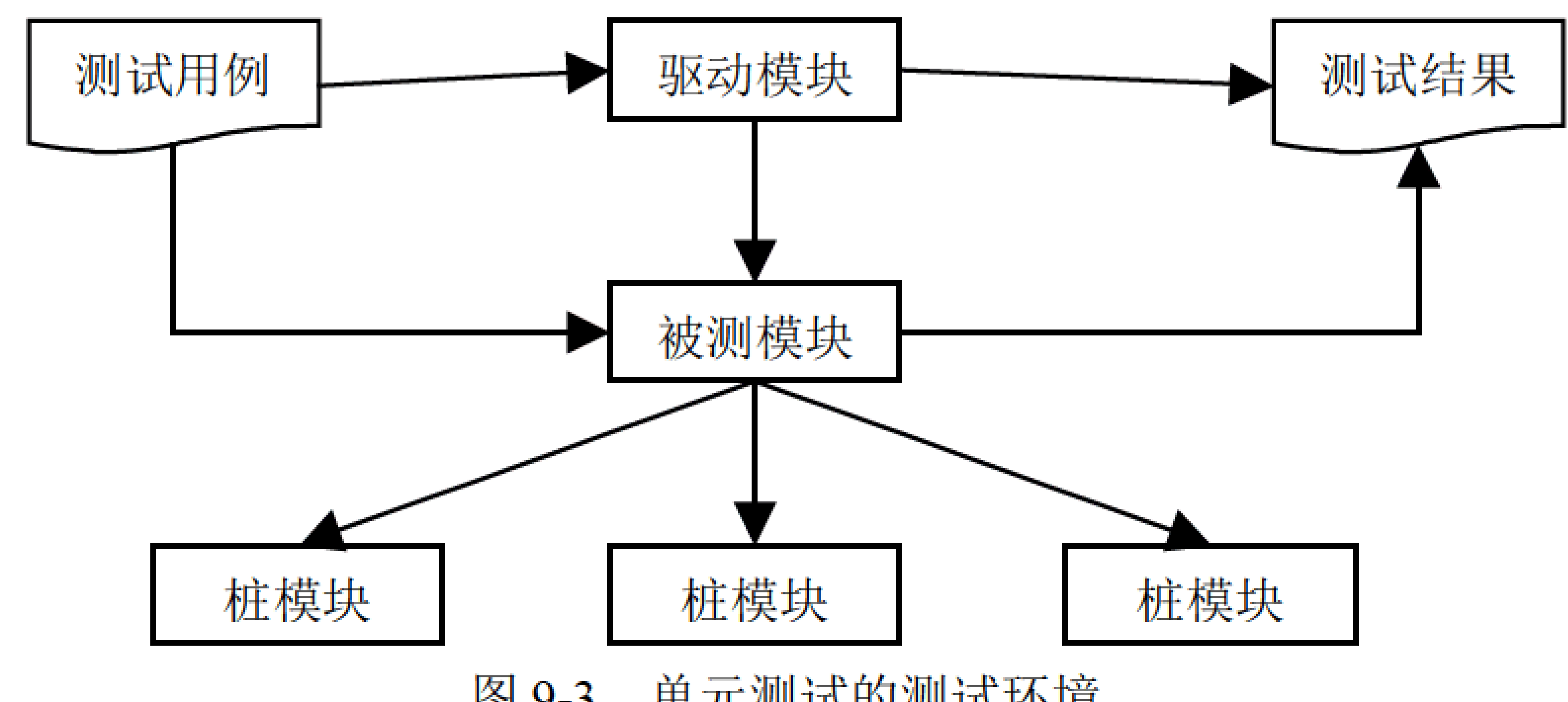


图 9-3　单元测试的测试环境

驱动模块和桩模块在软件开发结束后就不使用了，但是为了单元测试，两者都要进行开发，但是不需要与最终产品一起交付用户。因此，驱动模块和桩模块的设计要尽量简单，避免因其错误而干扰被测单元的运行及测试结果的判断。实际上许多程序单元不能用简单的驱动模块和桩模块进行充分的单元测试，完全的测试可以放到组装测试时再进行。

如果一个模块要完成多种功能，且以程序包或对象类的形式出现，例如 Ada 中的包、Modula 中的模块、C++中的类，这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互联测试。有人把这种情况特别称为模块测试，以区别单元测试。

9.2.2　单元测试步骤

在单元测试中，测试用例的运行环境的构建，以及测试用例的设计与测试集合的准备是至关重要的。因此，单元测试可按下列步骤进行：

第一步是要构造测试用例的运行环境，即确定测试用例运行的前提条件，明确被测模块或被测单元所需的程序环境(全局变量赋值或初始化实体)，启动测试驱动，设置桩，调用被测模块，设置预期输出条件判断，最后恢复环境(包括清除桩)。

第二步是设计黑盒测试用例，即接口测试用例。为此可以：

(1) 设计基本功能测试用例，证明被测单元至少在某种正常情况下能够运行了。

(2) 设计功能正面测试用例，找出被测单元对于设计要求的正确输入可能做出的不正确处理。

(3) 设计功能反面测试用例，找出被测单元对于设计要求的错误输入可能做出的不正确处理。

(4) 设计性能测试用例，找出被测单元对于设计要求的性能可能做不到的错误。

第三步是设计白盒测试用例，即覆盖测试用例，找出单元内部控制结构和数据使用可能存在的问题。

注意，在白盒测试执行期间，不要匆忙地删除所发现的死代码或冗余代码，因为这很可能导致错误的产生。因为在测试别人的代码时，很可能由于测试用例不够，或者没有对被测程序整体结构的把握，而出现错误理解。

9.3　单元测试工具与实践

9.3.1　单元测试工具 JUnit

1997 年，Erich Gamma 和 Kent Beck 为 Java 语言创建了一个简单但有效的单元测试框架，称作 JUnit。这项工作遵循 Kent Beck 在早些时候为 Smalltalk 创建的名为 SUnit 的测试框架的设计。

JUnit 是 SourceForge 上的一个开源软件，以 IBM 的 Common Public License 1.0 版授权协议发布。Common Public License 授权协议对商业用户是友好的，人们可以随同产品分发 JUnit，不需要很多官样文件，也没有很多限制。

JUnit 很快成了 Java 开发中单元测试框架的事实标准。事实上，基于 JUnit 的测试模型(即 xUnit)已正式成为各种语言的标准框架。ASP、C++、C#、Effel、Delphi、Perl、PHP、Python、REBOL、Smalltalk 和 Visual Basic 都已经有了 xUnit 框架。

那么 JUnit 的目标是什么呢？首先，回到开发的前提假设。假设一个程序不能自动测试，那么它就无法工作。但有更多的假设认为，如果开发人员保证程序能工作，那么它就会永远正常工作。与这个假设相比，我们的假设实在是太保守了。从这个观点出发，开发人员编写了代码，并进行了调试，但还不能说他的工作就完成了，他还必须编写测试脚本，以证明程序工作正常。然而，每个人都很忙，没有时间去进行测试工作。他们会说，我编写程序代码的时间都很紧，哪儿有时间去写测试代码呢？ 因此，首要的目标就是构建一个测试框架，在这个框架里，开发人员能编写测试代码。该框架要使用熟悉的工具，不用花很多精力就可以掌握。它还要消除不必要的代码，除了必需的测试代码外，消除重复劳动。

如果测试要做的仅仅是这些，那么在调试器中编写一个表达式就可以实现。但是，测试不仅仅是这些。虽然程序工作得很好，但这不够，因为不能保证集成后程序还能正常工作。因此，测试的第二个目标就是创建测试，并能保留这些测试，将来它们也是有价值的，其他人可以执行这些测试，并验证测试结果。如有可能，还要把不同人的测试收集在一起，一起执行，且不用担心它们之间会互相干扰。最后，还要能用已有的测试创建新的测试。每次创建新的测试是很花费代价的，框架能复用测试设置，执行不同的测试。

使用 JUnit 的好处有：

(1) 可以使测试代码与产品代码分开。

(2) 针对一个类的测试代码，通过较少的改动便可以将其应用于另一个类的测试。

(3) 易于集成到测试人员的构建过程中，JUnit 和 Ant 的结合可以实施增量开发。

(4) JUnit 公开源代码，可以进行二次开发。

(5) 可以方便地对 JUnit 进行扩展。

JUnit 测试的编写原则：

(1) 简化测试的编写，这种简化包括对测试框架的学习和实际测试单元的编写。

(2) 使测试单元保持持久性。

(3) 可以利用既有的测试来编写相关的测试。

JUnit 的特征：

(1) 使用断言方法来判断期望值和实际值的差异，返回 Boolean 值。

(2) 测试驱动设备使用共同的初始化变量或实例。

(3) 测试包结构便于组织和集成运行。

(4) 支持图形交互模式和文本交互模式。

JUnit 共有 7 个包，核心的包就是 junit.framework 和 junit.runner。framework 包负责整个测试对象的构架，runner 包负责测试驱动。

JUnit 有 4 个重要的类：TestSuite、TestCase、TestResult 和 TestRunner。前三个类属于 framework 包，后一个类在不同的环境下是不同的。如果使用的是文本测试环境，这时用的就是 junit.textui.TestRunner。各个类的职责如下：

(1) TestResult：负责收集 TestCase 执行的结果。它将结果分为两类：客户可预测的 Failure 和没有预测的 Error。同时负责将测试结果转发到 TestListener(该接口由 TestRunner 继承)进行处理。

(2) TestRunner：客户对象调用的起点，负责对整个测试流程进行跟踪，能够显示返回的测试结果，并且报告测试的进度。

(3) TestSuite：负责包装和运行所有的 TestCase。

(4) TestCase：客户测试类所要继承的类，负责测试时对客户类进行初始化以及测试方法调用。

另外还有两个重要的接口：Test 和 TestListener。

(1) Test：包含两个方法——run( )和 countTestCases( )，用于对测试动作的特征进行提取。

(2) TestListener：包含 4 个方法——addError( )、addFailure( )、startTest( )和 endTest( )，用于对测试结果的处理以及测试驱动过程中动作特征的提取。

JUnit 框架的组成：

(1) 对测试目标进行测试的方法与过程集合，可称为测试用例(TestCase)。

(2) 测试用例的集合，可容纳多个测试用例(TestCase)，可称为测试包(TestSuite)。

(3) 测试结果的描述与记录(TestResult)。

(4) 测试过程中的事件监听者(TestListener)。

(5) 每一个测试方法所发生的与预期不一致的状况的描述，可称为测试失败元素(TestFailure)。

(6) JUnit 框架中的出错异常(AssertionFailedError)。

(7) JUnit 框架是一种典型的 Composite 模式：TestSuite 可以容纳任何派生自 Test 类的对象；当调用 TestSuite 对象的 run( )方法时，会遍历自己容纳的对象，逐个调用它们的 run( )方法。

典型的使用 JUnit 的方法就是继承 TestCase 类，然后重载它的一些重要方法：setUp( )、tearDown( )和 runTest( )(这些都是可选的)。最后将这些客户对象组装到一个 TestSuite 对象中，交由 junit.textui.TestRunner.run(用例集)驱动。

1. JUnit 单元测试环境的建立

由于 JUnit 越来越流行，现在 Java 的很多 IDE 都会集成 JUnit 插件，使用起来也非常方便。虽然现在绝大部分开发人员都使用 IDE 工具来开发程序，但是为了能够让更多的人体验 JUnit 而不必去下载体积庞大的 IDE 工具，本节将分两种情况介绍 JUnit 单元测试环境的建立。一种是最直接的方式：配置 JUnit，通过命令行来建立。另一种则是使用 Eclipse 中的 JUnit 插件来建立。

注意：

使用 JUnit 和 IDE 的前提是系统中已安装 Java 的 JDK(Java Develope Kit)，并正确加入了系统环境变量。

1) 独立 JUnit 单元测试环境的建立

(1) 从 www.junit.org(JUnit 官方网站)下载最新的 JUnit 包(这里使用的是 JUnit 4.1.1 版本)，如图 9-4 所示。

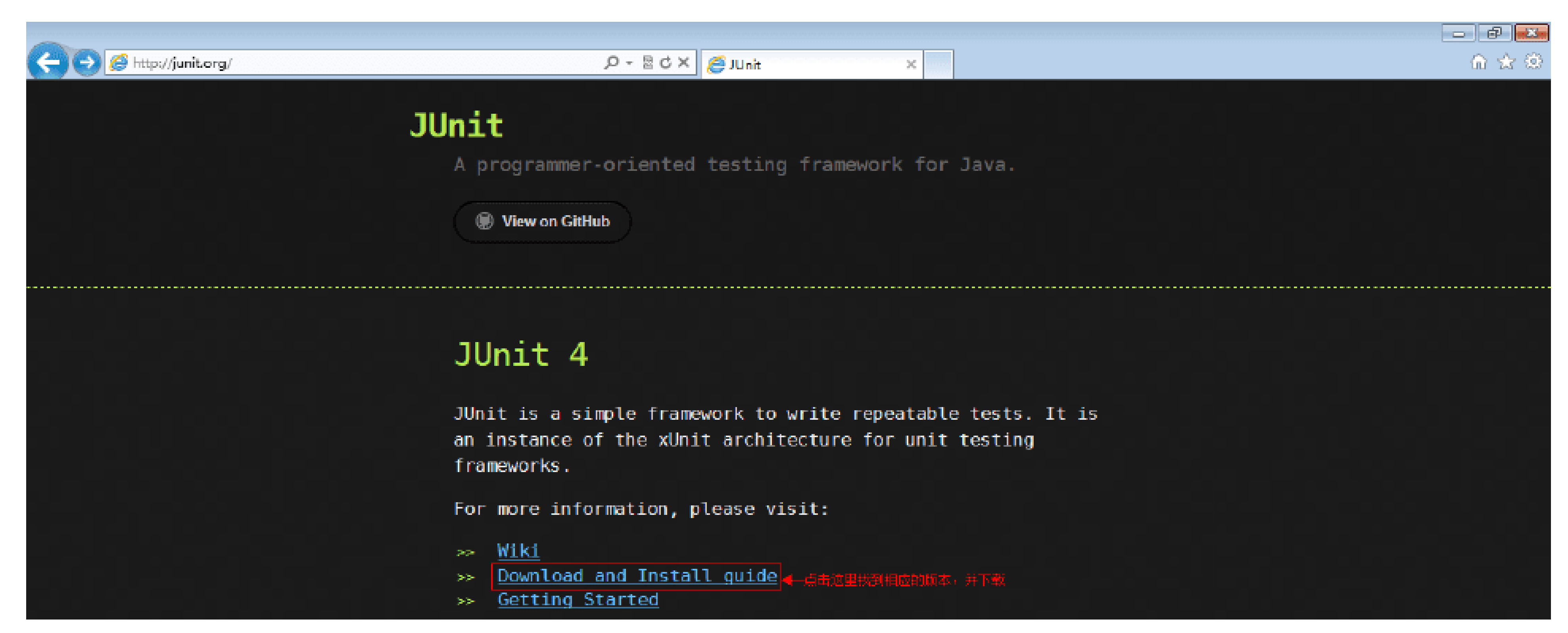


图 9-4　JUnit 官方网站下载页面

(2) 将 JUnit 的压缩包解压到硬盘上(这里是解压到 C 盘)，如图 9-5 所示。

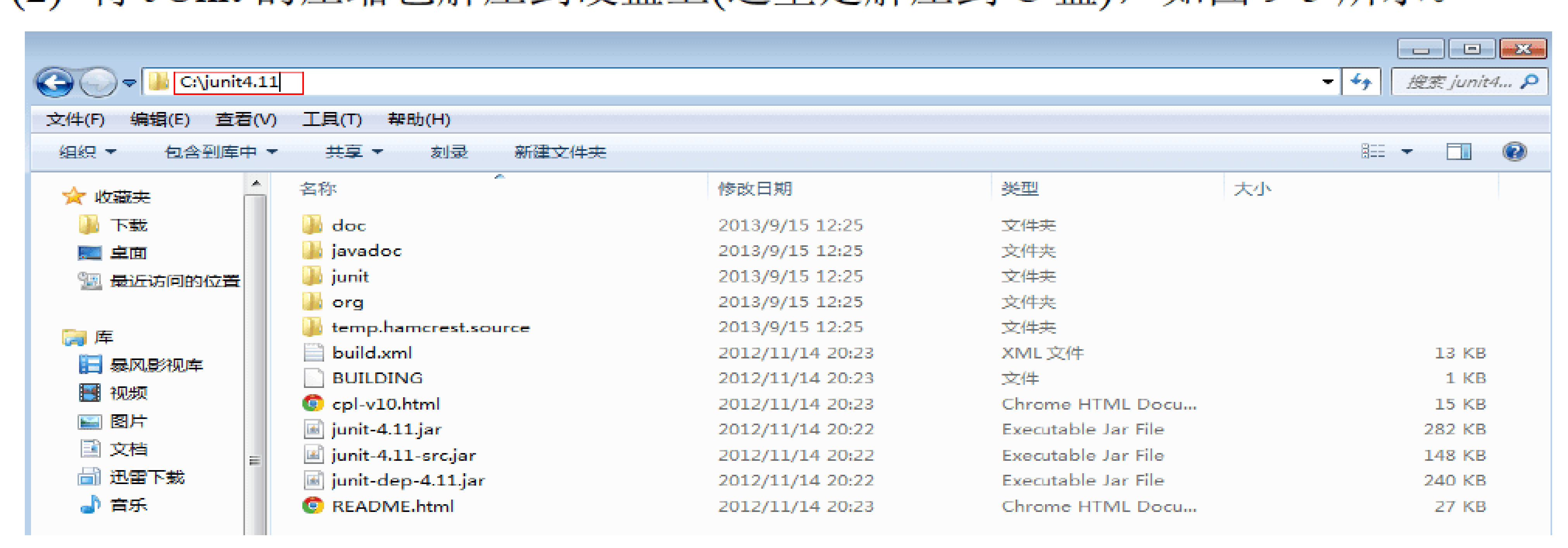
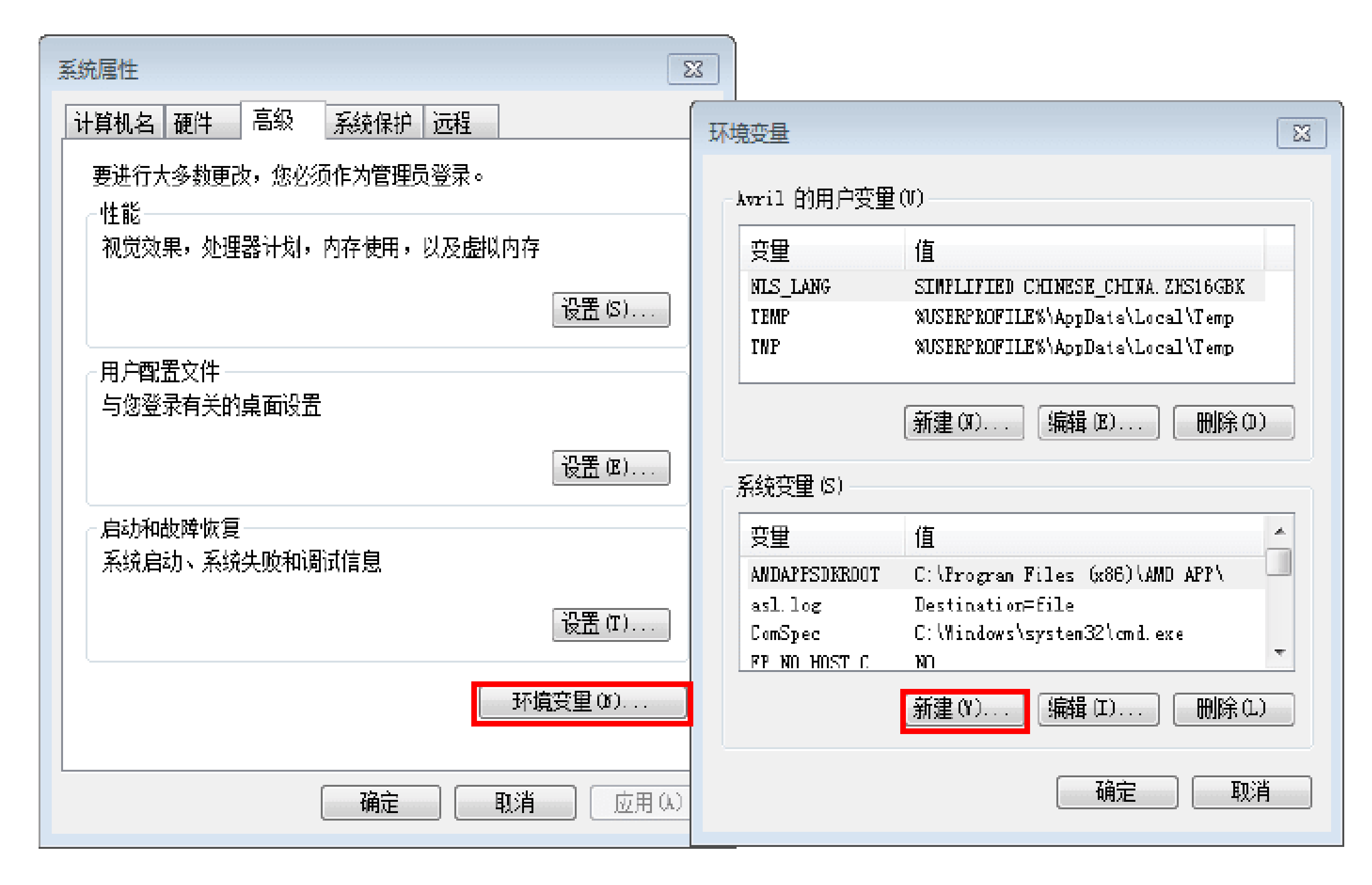


图 9-5　将 JUnit 解压到本地硬盘

(3) 将 JUnit 的 jar 包(这里的示例为 junit.jar，JUnit 版本不同，该 jar 包的名字也会不同)添加到环境变量的 classpath(默认的 class 文件路径变量)中。

(4) 右击“我的电脑”，依次选择“属性”|“高级”|“环境变量”。如果操作系统的环境变量中没有 classpath 这个变量，则需要手动添加一个。单击“系统变量”下的“新建”按钮，如图 9-6 所示，在打开的对话框中按照图 9-7 所示的信息填写。



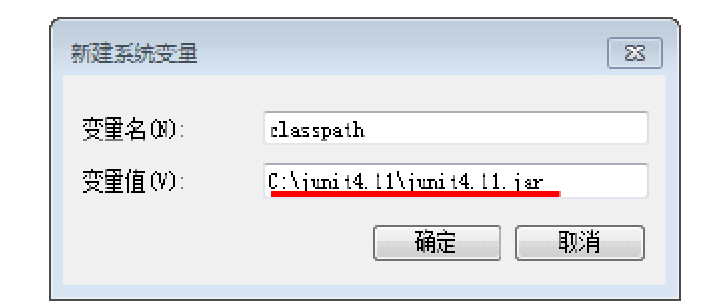


图 9-6　“环境变量”对话框　　　图 9-7　设置环境变量

注意：

图 9-7 中下画线标注的部分会根据 JUnit 安装路径和版本的不同而变化，请做相应更改。

(5) 测试安装是否成功。

有以下三种方式可以用来进行测试：

① 批处理文本方式，在 cmd 命令行中输入如图 9-8 所示的命令。



图 9-8　命令行测试

② AWT 图形测试运行方式，在 cmd 命令行中输入如图 9-9(a)所示的命令。

③ 基于 Swing 的图形测试方式，在 cmd 命令行中输入如图 9-9(b)所示的命令。

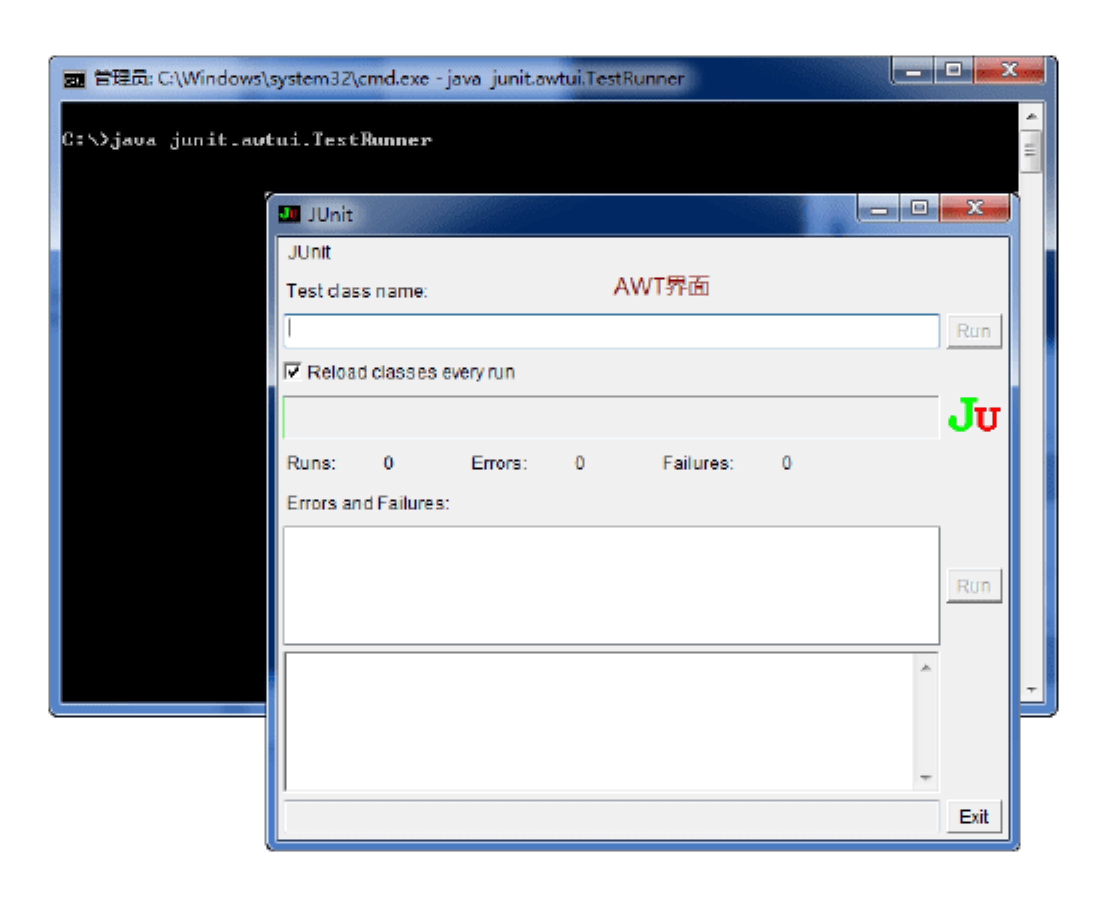


图 9-9(a)　AWT 图形测试

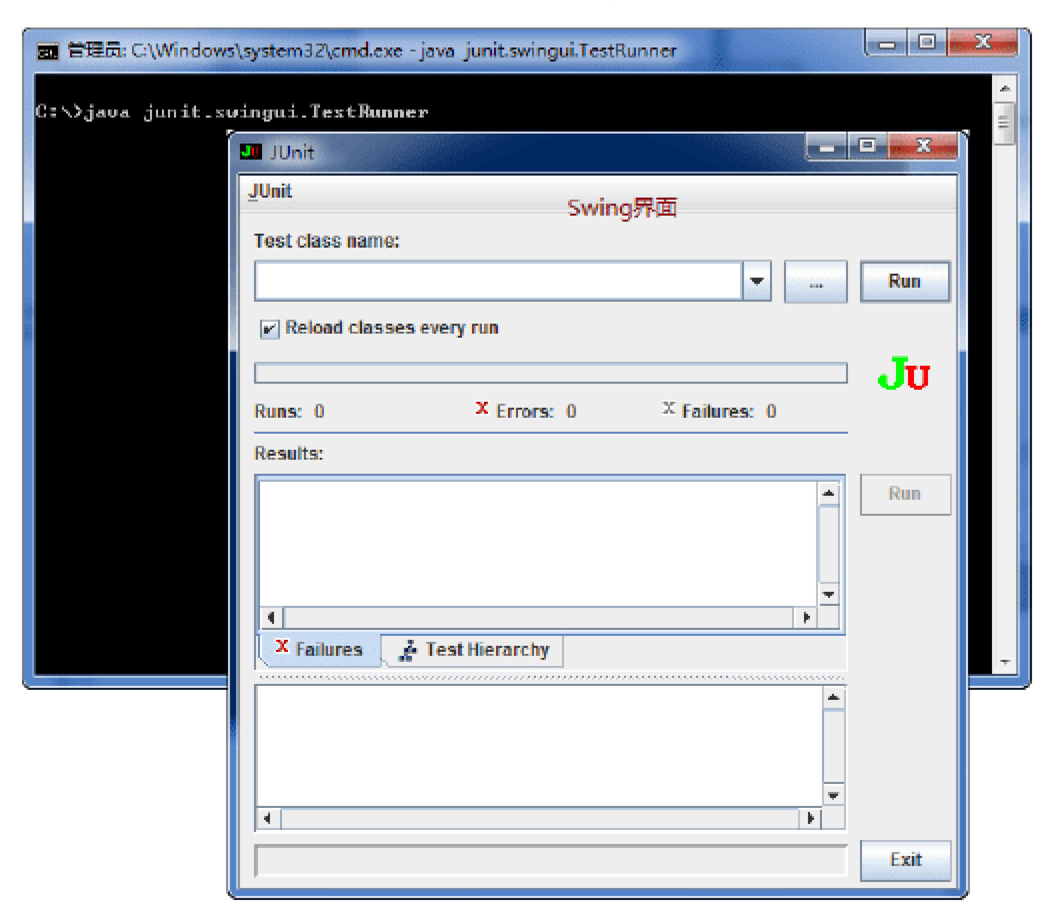


图 9-9(b)　Swing 图形测试

如果上述命令都运行无误，则安装成功。

2)　Eclipse 中的 JUnit 插件

由于 Eclipse 中已经集成了 JUnit 插件，因此可以直接建立测试用例来测试代码。即使使用的 Eclipse 中没有集成 JUnit，也没有关系。不需要特别的配置，只需要在工程属性中引入 JUnit 的 jar 包即可使用。

(1) Eclipse 中包含 JUnit 4 插件

此时，只需要在工程中应用即可，应用方法如下。

① 右击 Java 工程，选择“属性”命令，打开如图 9-10 所示的对话框，并选择 Java Build Path。

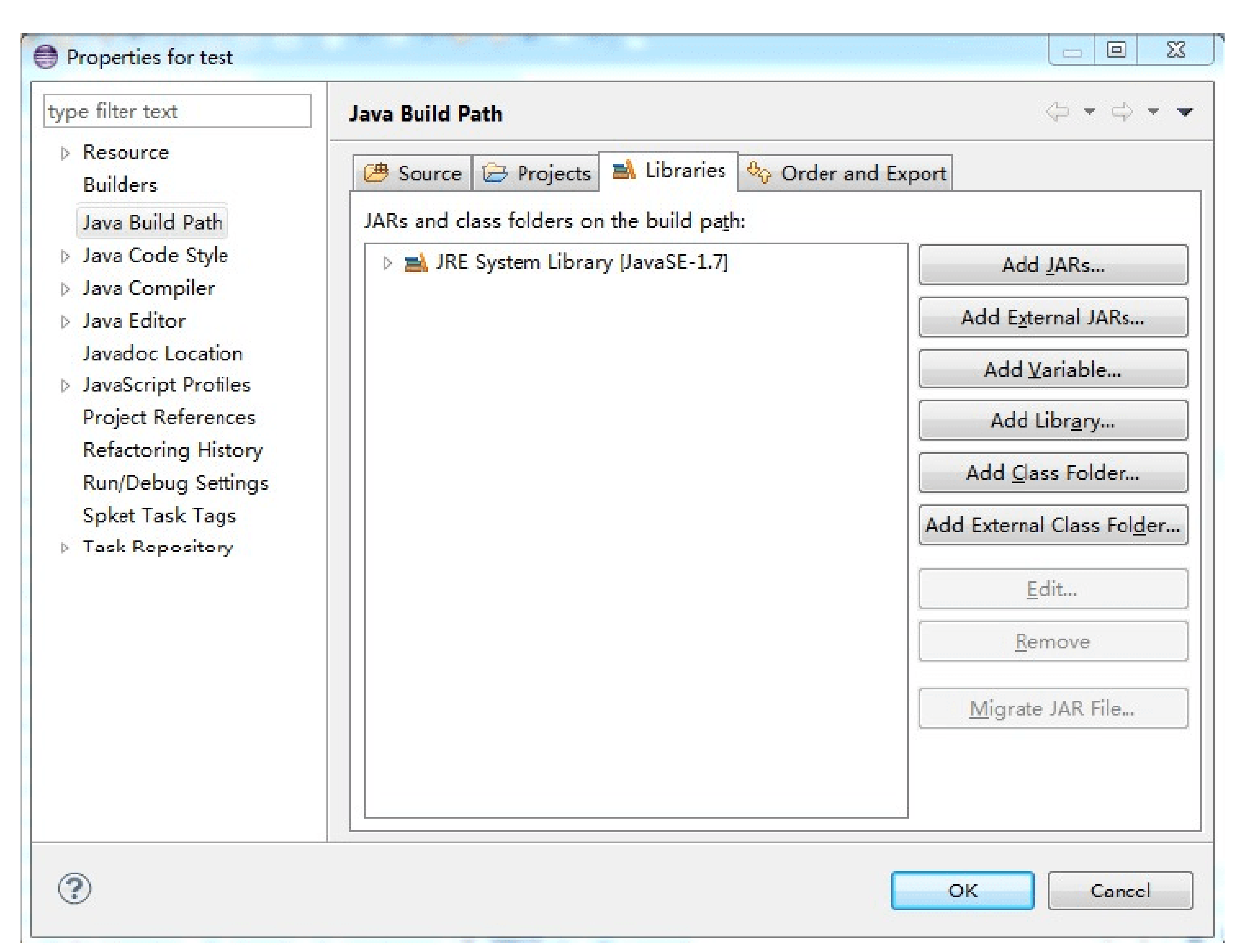


图 9-10　选择 Java Build Path

② 在 Libraries 选项卡中单击 Add Library 按钮，选择 JUnit 4，如图 9-11 所示。

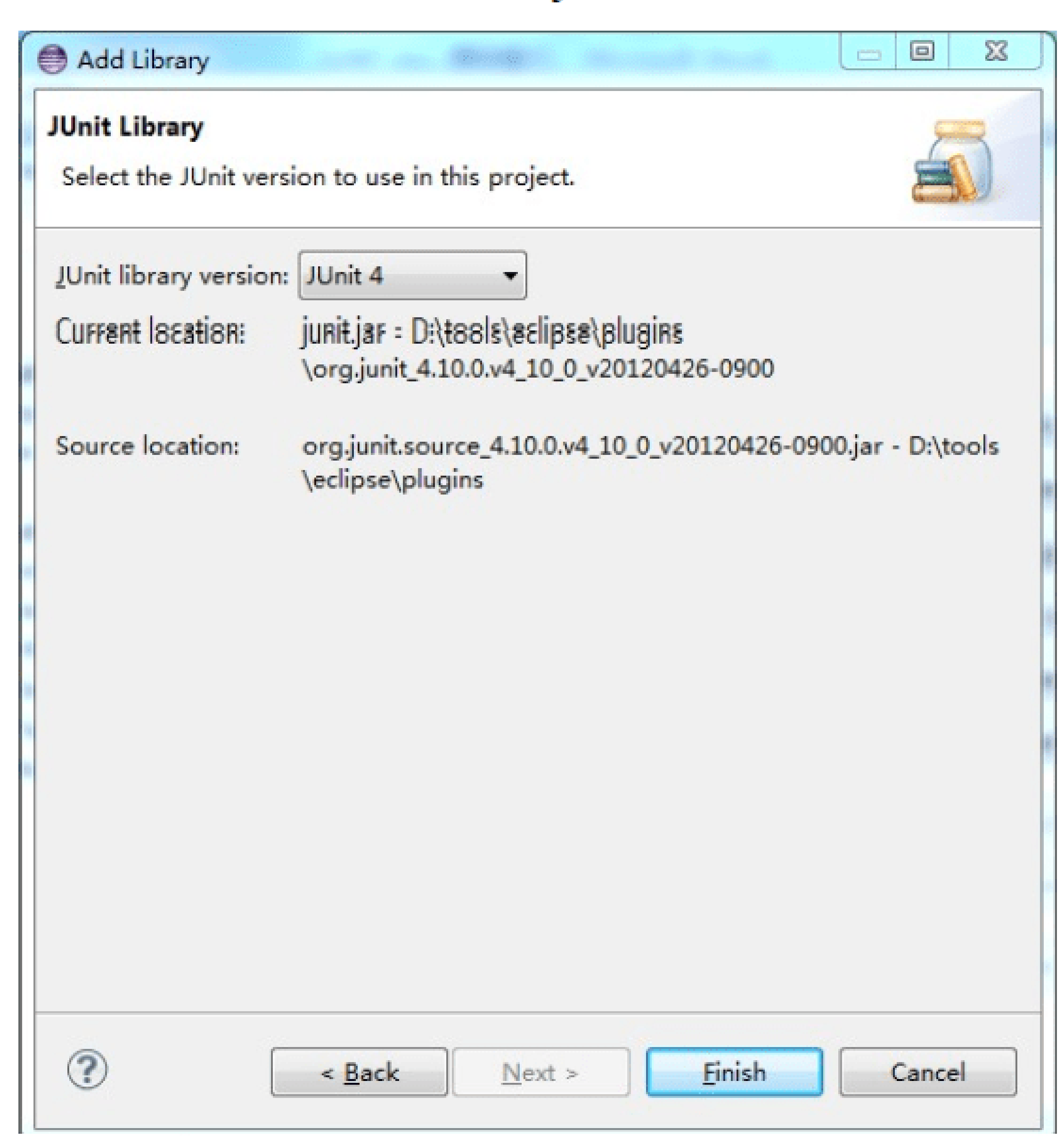


图 9-11　选择 JUnit 4

③ 单击 Finish 按钮，在工程目录下即可看到 JUnit 4 包，如图 9-12 所示。

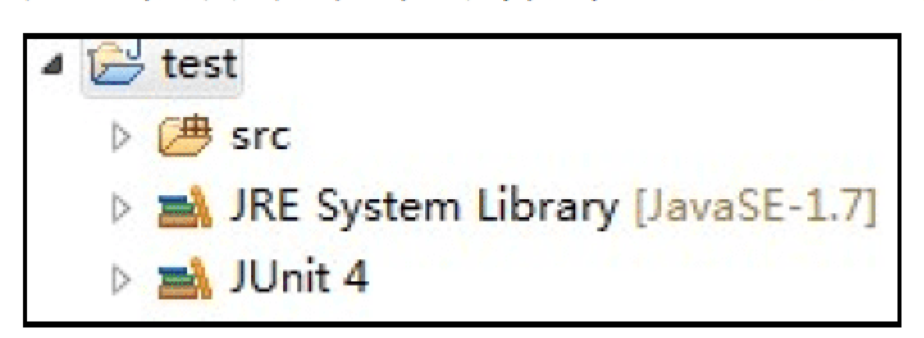


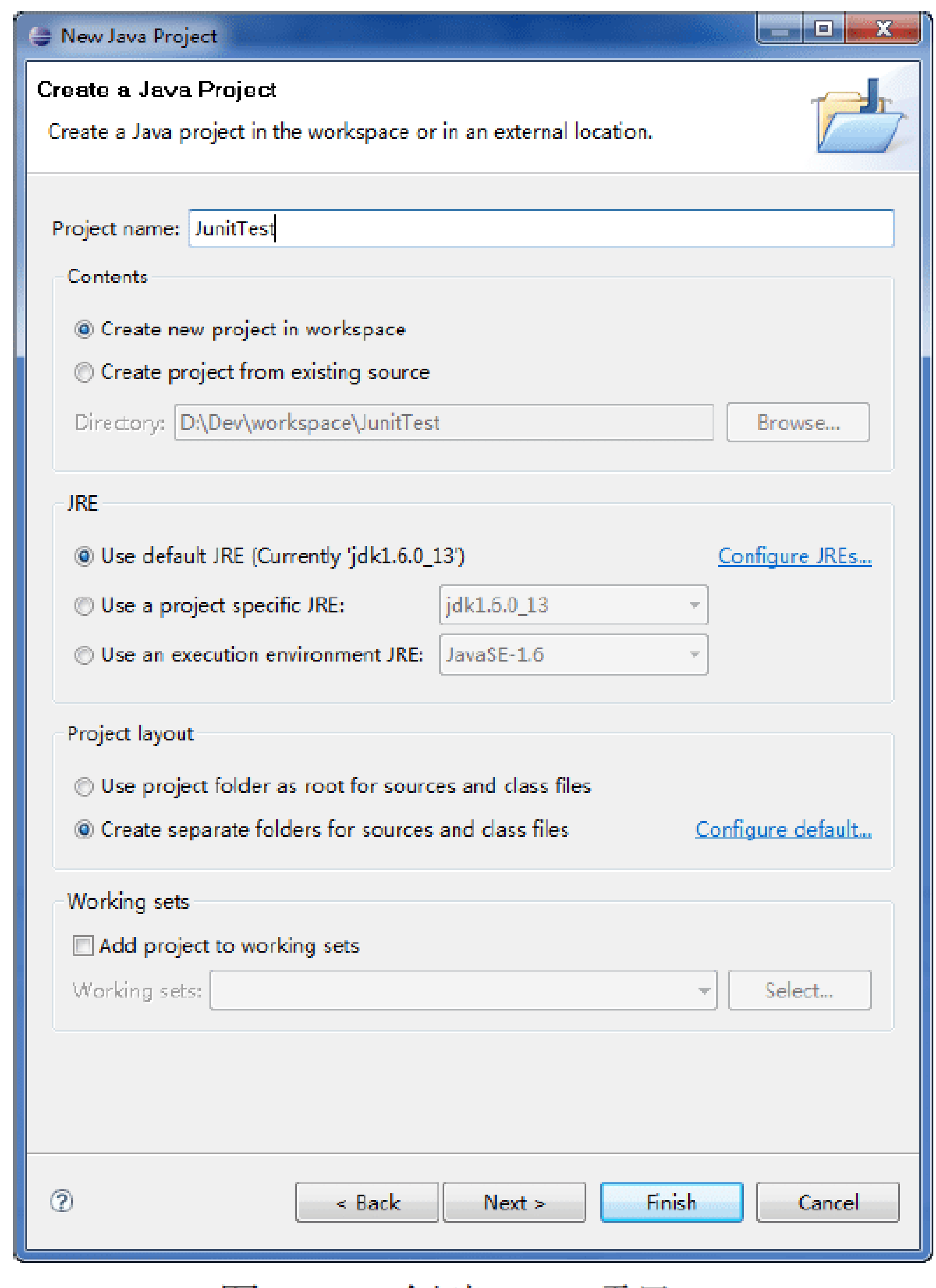
图 9-12　工程目录下的 JUnit 4 包

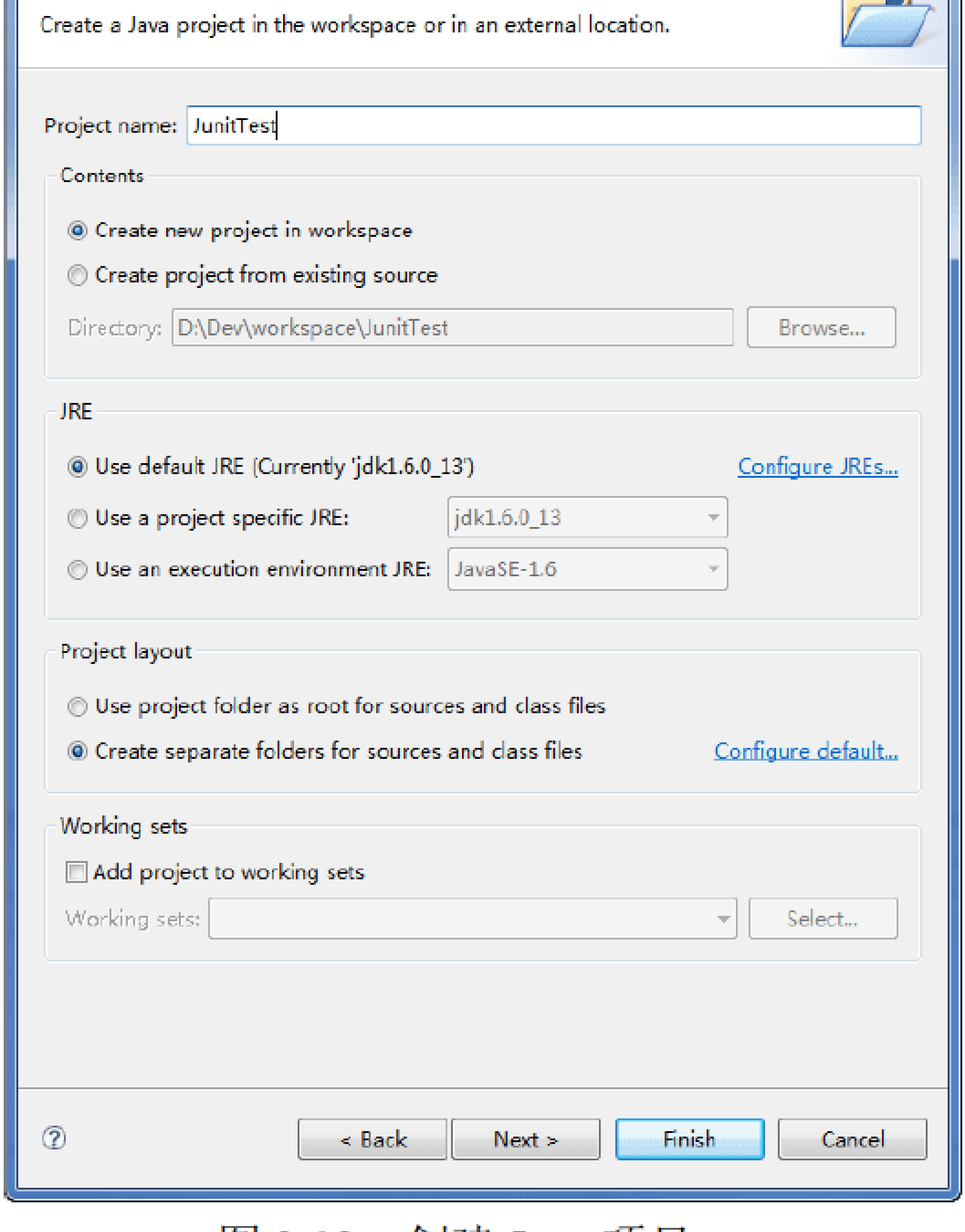
(2) Eclipse 中未包含 JUnit 4 插件

如果 Eclipse 中没有集成 JUnit 插件，按以下步骤就可以轻松集成这个强大的测试框架：

① 下载 JUnit 的集成包，并将其解压缩到硬盘上(方法与前面介绍的一样，这里不再赘述)。然后在 Eclipse 中创建一个名为 JunitTest 的 Java 项目，如图 9-13 所示。

② 在 Package Explorer 中，右击刚才创建的项目的名称，在弹出的菜单中选择 Properties 命令，如图 9-14 所示。





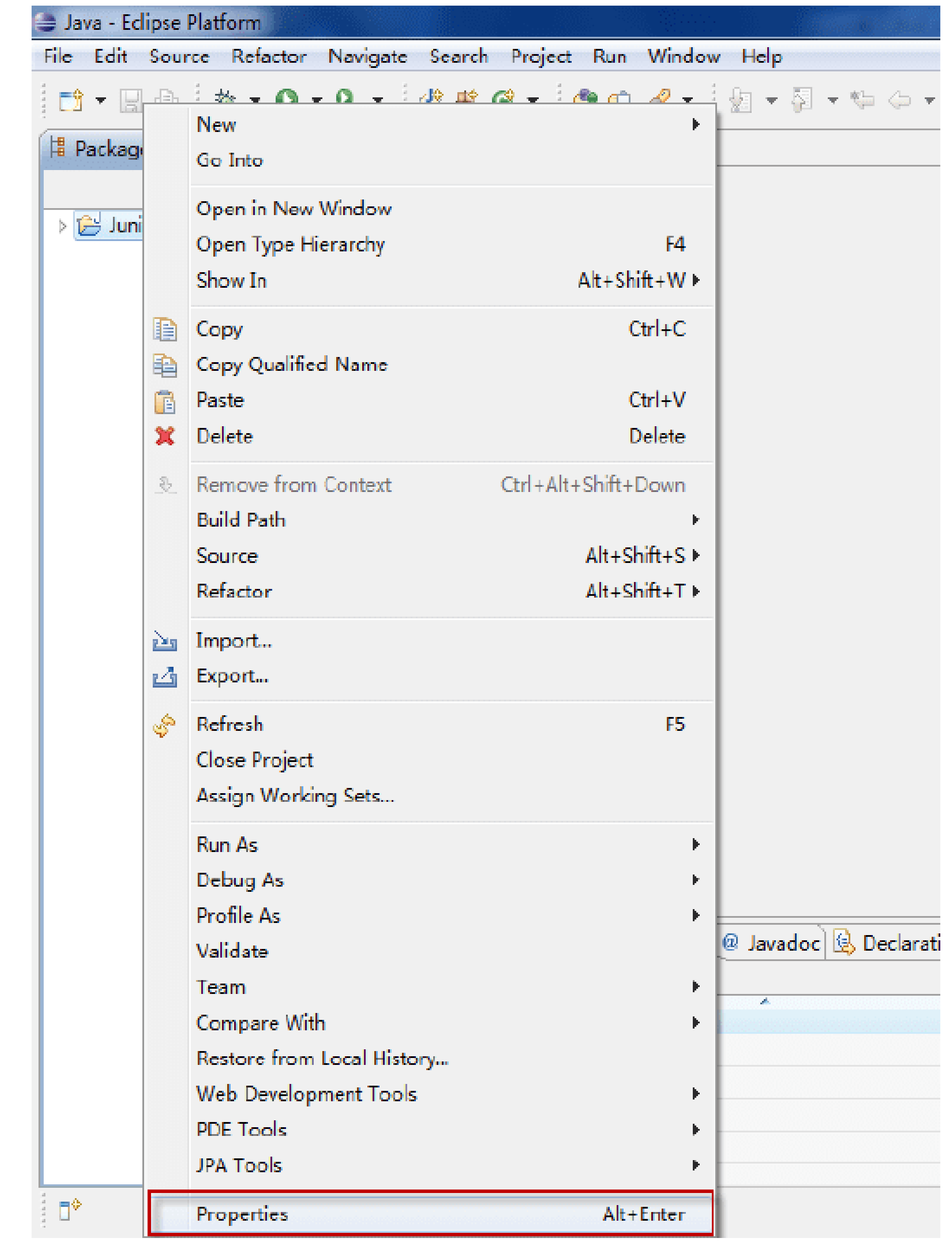


图 9-13　创建 Java 项目　　　　　　图 9-14　设置 Java 项目

③ 依次选择 Java Build Path|Libraries，单击 Add External JARs 按钮，导航至 JUnit 解压缩的目录，选择 junit.jar 包，打开即可，如图 9-15 所示。

④ 随便建立一个 Java 文件，右击这个文件，在菜单中选择 New 命令，这时候里面会有 JUnit Test Case 选项，单击它，就可以创建 JUnit 测试用例了，如图 9-16 所示。

JUnit 会根据选择的文件，自动把测试用例需要的参数填充完整。当然，如果不是根据文件创建的测试用例，而是完全自己手工编写，那么需要注意把所需的参数填全。

如果 Eclipse 中已经集成了 JUnit，那么根据步骤④中的介绍直接使用 JUnit 即可。

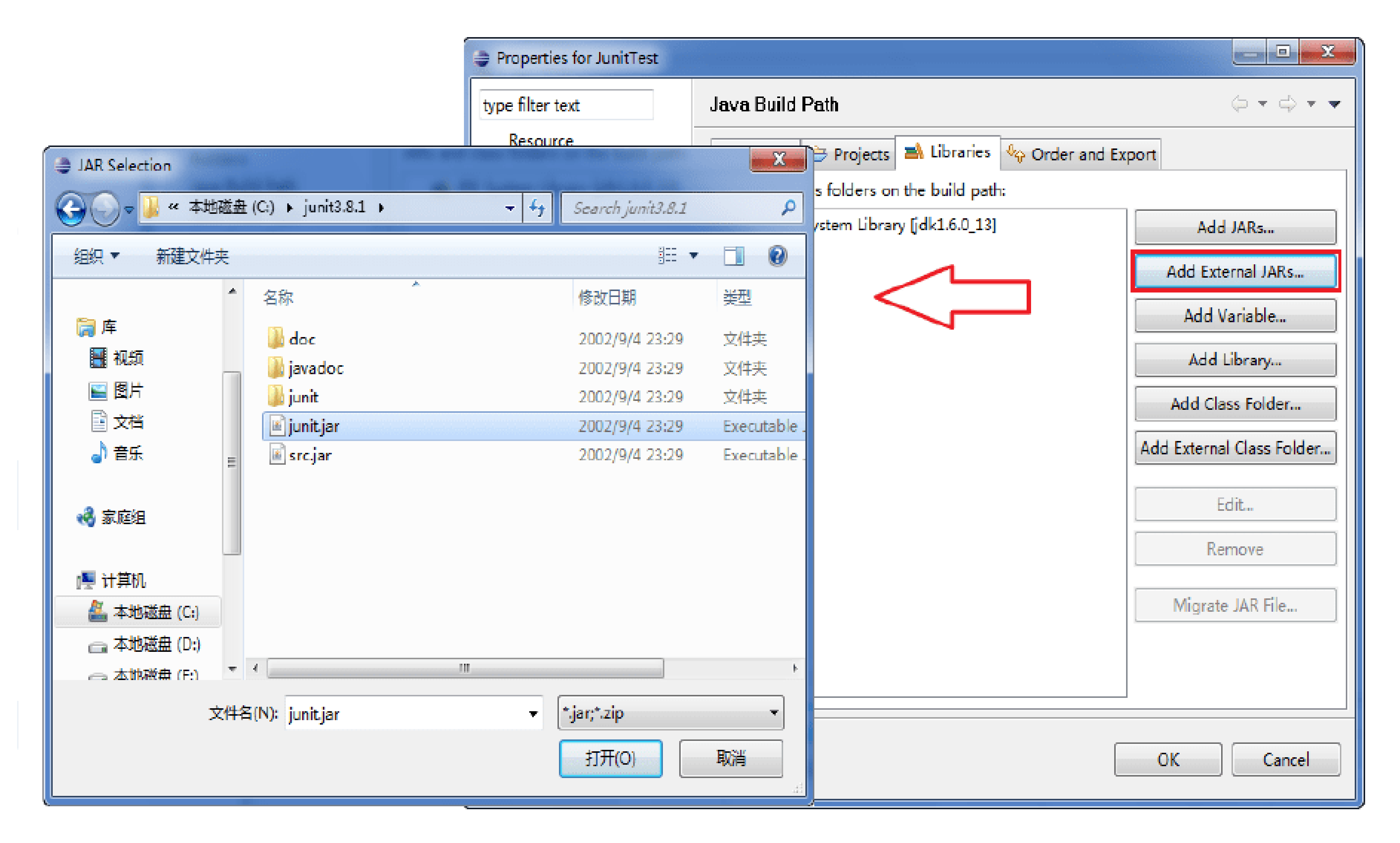


图 9-15　添加 JUnit 支持

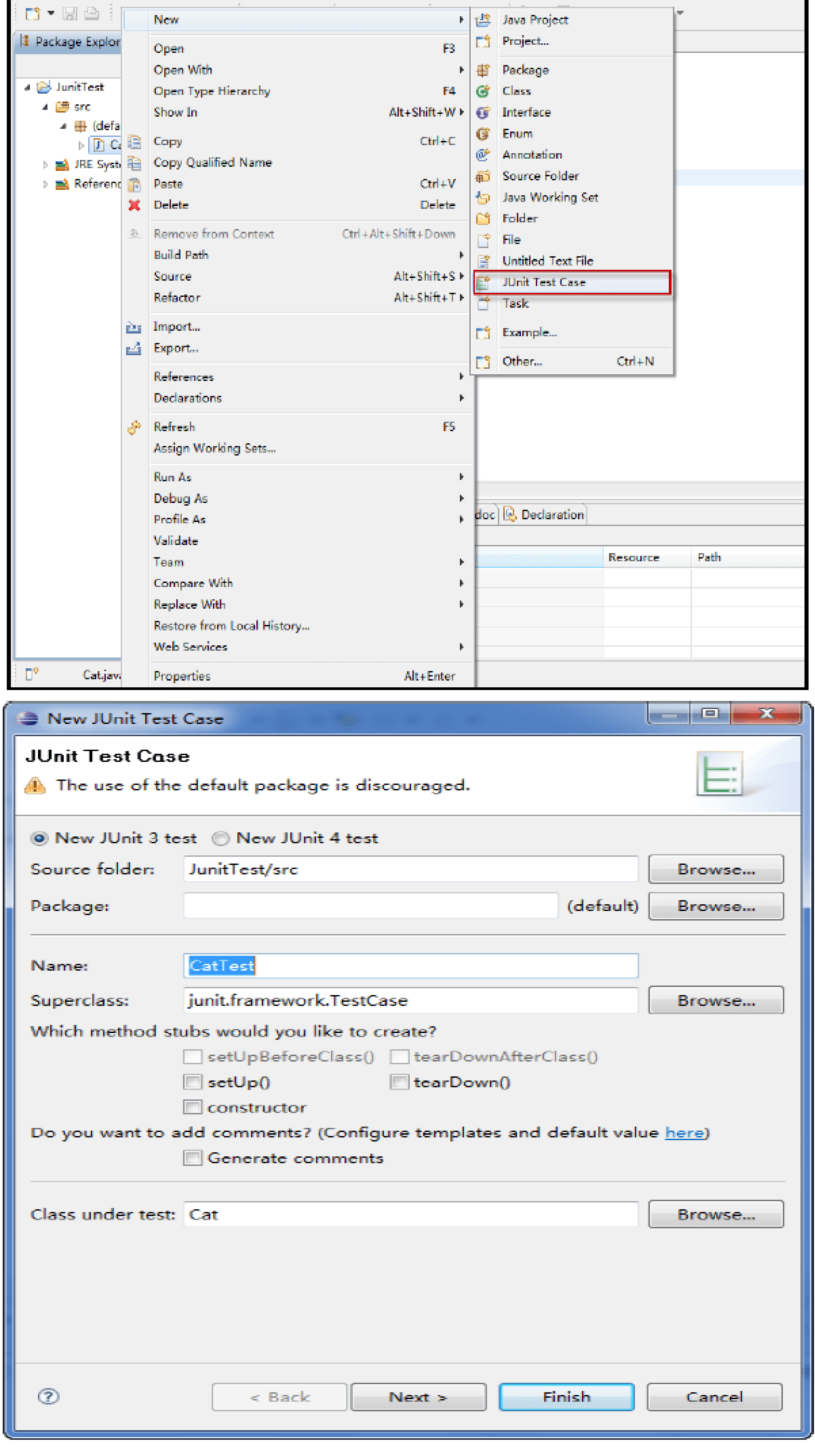


图 9-16　创建 JUnit 测试用例

2. JUnit 单元测试方法

接下来，可能会先编写测试代码，再编写工作代码；或者相反，先编写工作代码，再编写测试代码。按照 XP 编程开发方法，应先编写测试代码，再编写工作代码。因为这样可以在编写工作代码时清晰地了解工作类的行为。

需要注意，编写那些能通过的测试的测试代码，意义不是十分突出，而编写那些能帮助发现 bug 的测试代码才有价值。此外，测试代码还应该对工作代码进行全面的测试。

根据上面介绍 JUnit 测试环境搭建的内容，这里也分两种情况介绍测试方法。

1) 独立 JUnit 应用

(1) 创建一个简单的 Java 类，存放于 C 盘 JunitTest 目录下。该 Java 类的代码为：

```
public class Cat {
        public String getName(){
            return "Hello Kitty";
        }
 }
```

(2) 创建该类的测试类，存放于同一个目录下。该测试类的代码为：

```
import junit.framework.*;
 public class TestCat extends TestCase {
 protected String expectedLegs;
 protected Cat myCat;
     public TestCat(String name) {
          super(name);
     }
     // 设定了进行初始化的任务
     protected void setUp() {
          expectedLegs = Hello Kitty;
          myCat = new Cat();
     }
     // 这是一个很特殊的静态方法。JUnit的TestRunner会调用suite( )方法来确定有多少个测试可以执行
     public static Test suite() {
     return new TestSuite(TestCat.class);
     }
     // 对预期的值和 myCat.getLegs( )返回的值进行比较，并打印比较结果
     public void testGetLegs() {
          assertEquals(expectedLegs, myCat.getName());
     }
}
```

(3) 编译该测试类。

(4) 输入如图 9-17 所示的命令来执行 JUnit 测试。

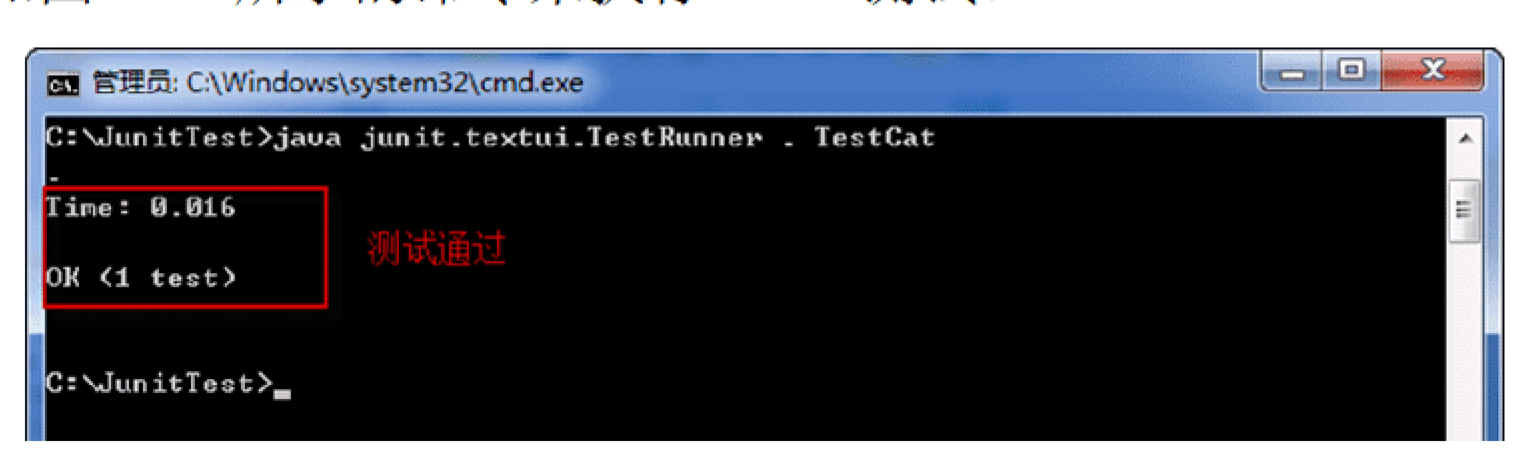


图 9-17　执行 JUnit 测试用例(命令行模式)

如果想启动 Swing 或 AWT 的 JUnit 界面来执行测试，则需要输入命令 java junit.swingui.TestRunner.TestCat 或 java junit.awtui.TestRunner.TestCat，如图 9-18 所示。

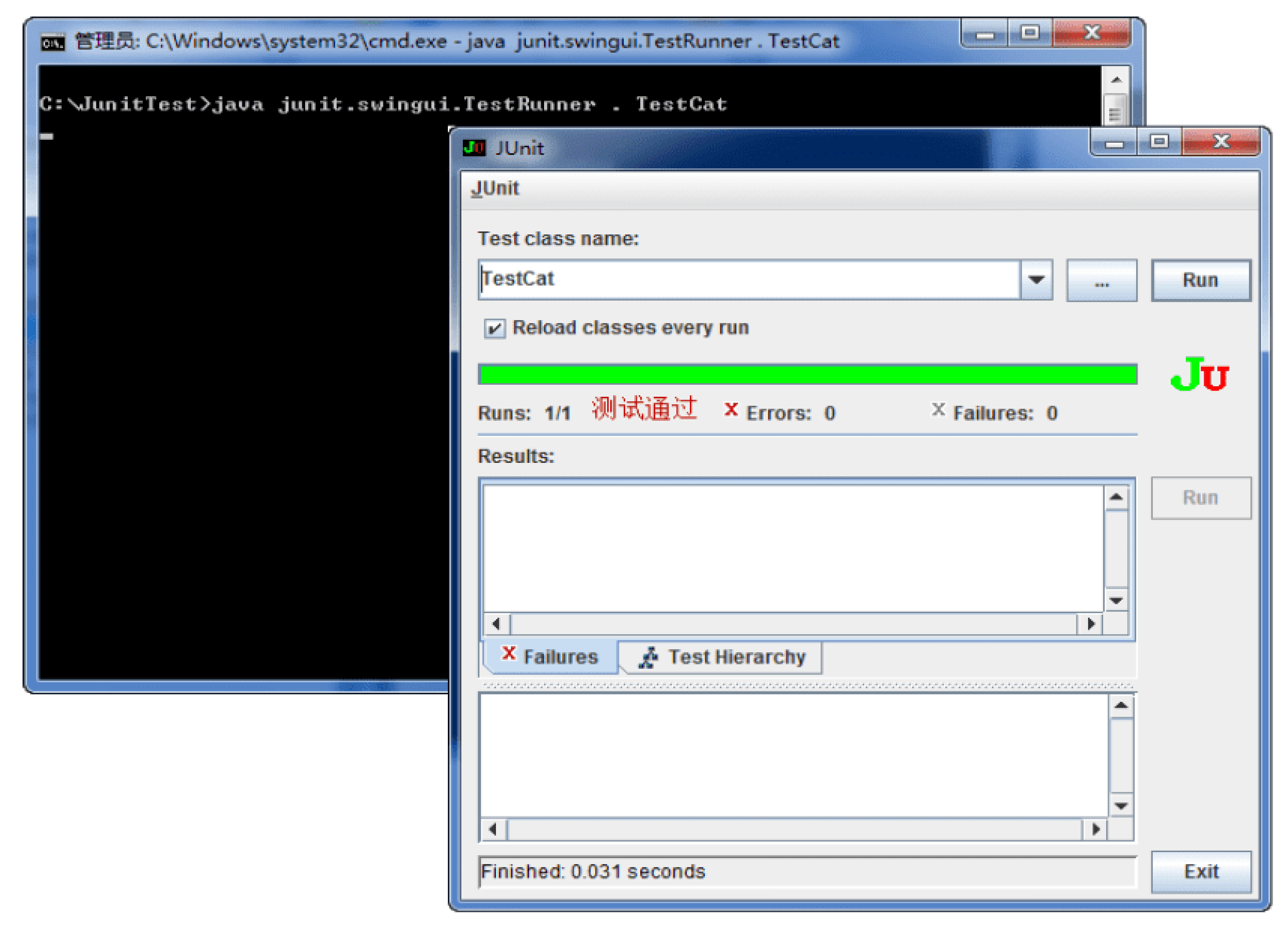


图 9-18　执行 JUnit 测试用例(Swing 界面模式)

2) Eclipse 中的 JUnit 应用

(1) 创建一个新的 Java 项目，然后创建一个简单的 Java 类，如图 9-19 所示。

```
Cat.java   TestCat.java

public class Cat {

    public String getName(){
        return "Hello Kitty";
    }
}
```

图 9-19　创建 Cat 类

(2) 按照前面介绍的方式，为 Cat 类创建一个 JUnit 测试类，如图 9-20 所示。注意图 9-20 中下画线标注的代码被修改过，这是为了测试 JUnit 测试不通过的情况。

```
Cat.java   TestCat.java

import junit.framework.*;

public class TestCat extends TestCase {

    protected int expectedLegs;

    protected Cat myCat;

    public TestCat(String name) {
        super(name);
    }

    // 设定了进行初始化的任务
    protected void setUp() {
        expectedLegs = 4;      实际应为String类型的"Hello Kitty"
        myCat = new Cat();
    }

    // 是一个很特殊的静态方法。JUnit的TestRunner会调用suite方法来确定有多少个测试可以
    public static Test suite() {

        return new TestSuite(TestCat.class);
    }

    // 对预期的值和myCat.getLegs()返回的值进行比较，并打印比较的结果
    public void testGetLegs() {
```

图 9-20　Cat 类的测试类 TestCat

(3) 依次选择 Eclipse 菜单栏中的 Run|Run as|JUnit Test 命令，执行 JUnit 测试。由于修改了断言中的内容，因此 JUnit 测试应该是通不过的，如图 9-21 所示。

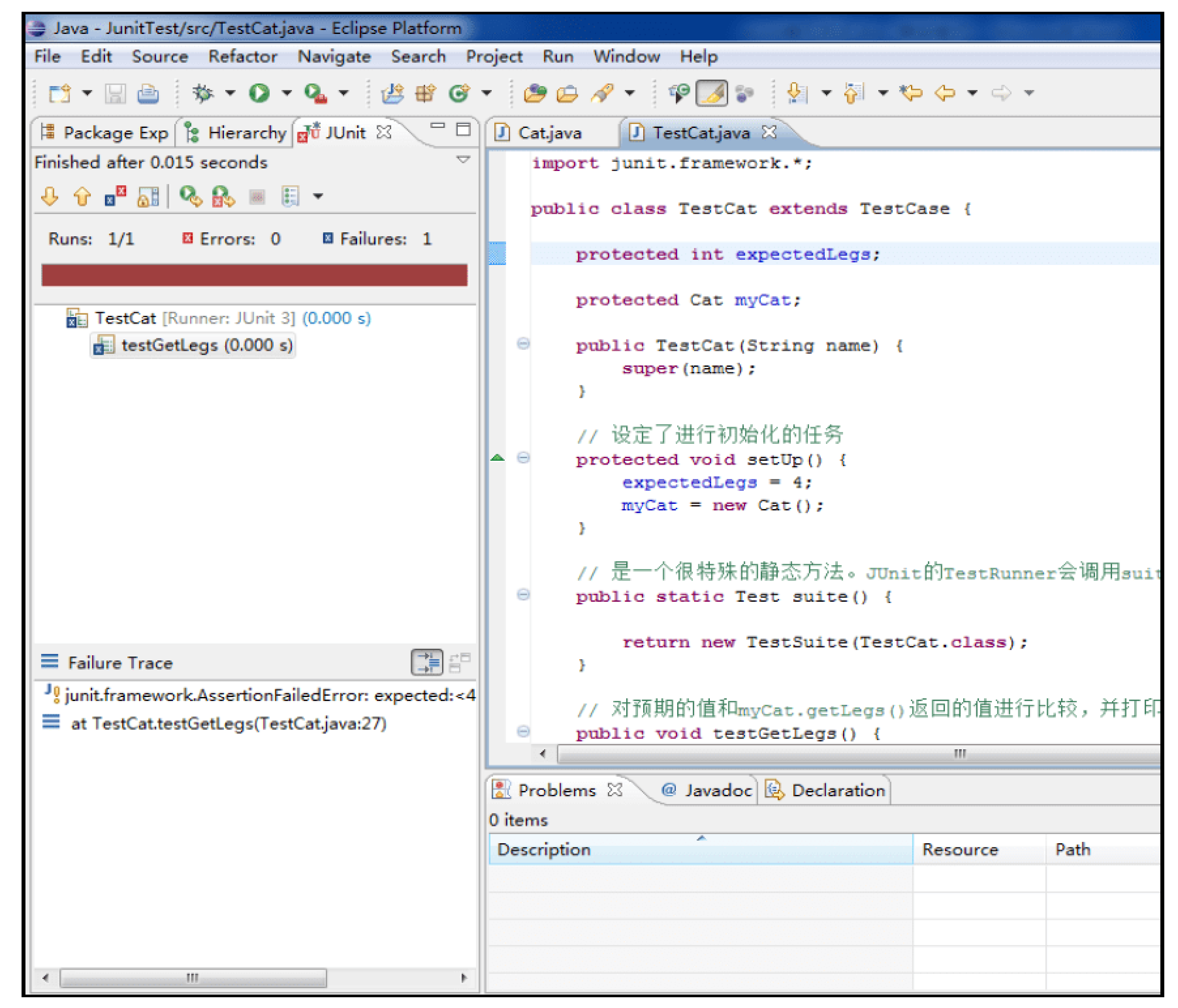


图 9-21　未通过的 JUnit 测

把 TestCat 类中的代码修改为正确的，再执行 JUnit 测试，结果可以通过，如图 9-22 所示。

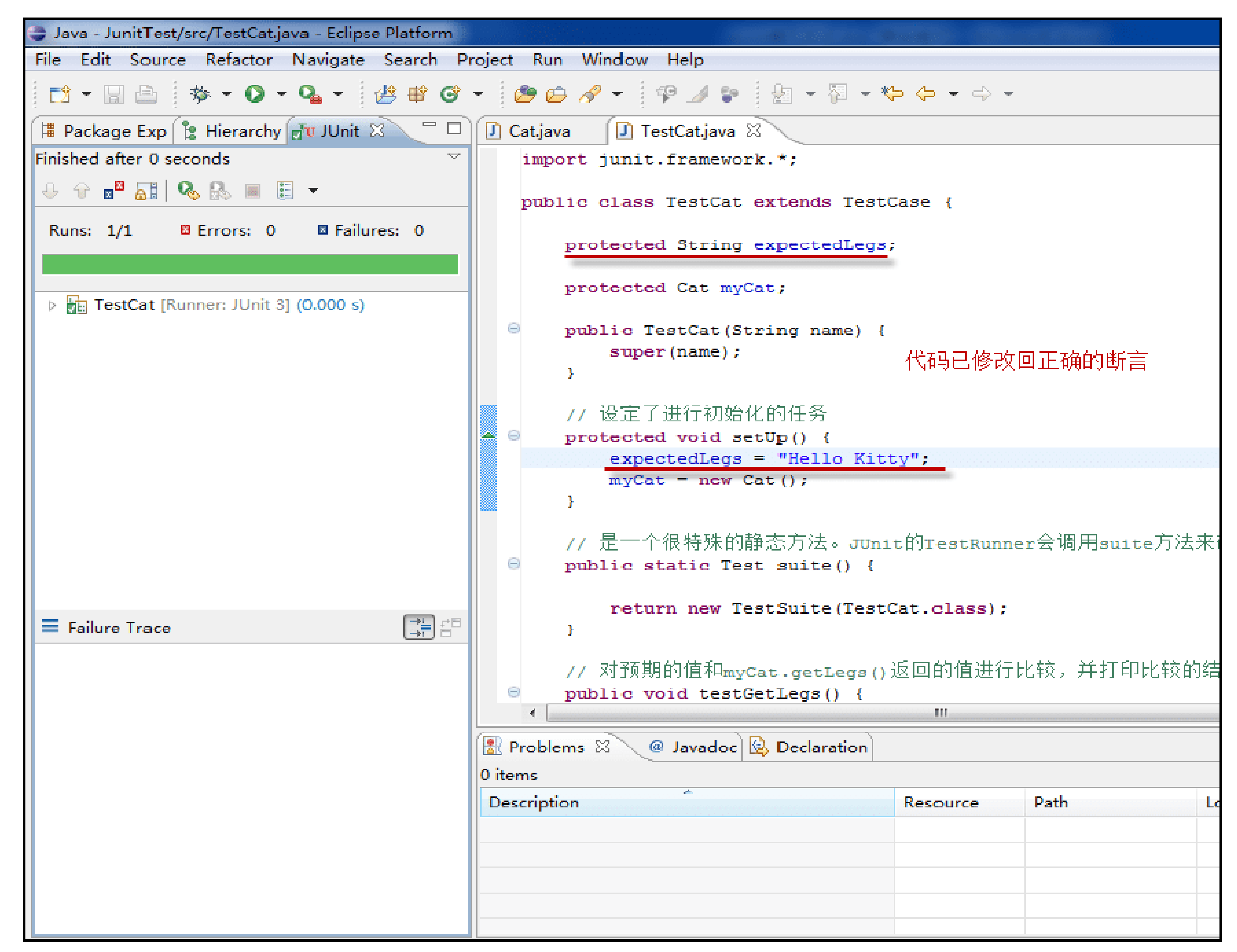


图 9-22　通过的 JUnit 测试

3. JUnit 单元测试应用举例

1) 以添加新课程为例来说明 JUnit 测试过程

在高校学生选课管理系统的课程管理模块中，单击“增加新课程”按钮后，可以进行新课程的添加，具体列表如图 9-23 所示：

(1) 若选择专业，需要单击“选择专业”下拉列表，从下拉列表中选择现有专业名称。

(2) 若填写课程名称、上课时间、上课地点、课程学分、课程介绍、授课教师、教师介绍等具体信息，需要在相应文本框中输入相关信息。

(3) 新课程信息填写完成后，可以通过统计信息模块查看已添加的新课程。

当前位置：课程管理->添加新课程

选择专业：	2005届2年制美术装潢设计专业	请选择专业
课程名称：		输入课程名称
上课时间：		输入上课时间
上课地点：		输入上课地点
课程学分：		输入课程学分
课程介绍：		输入课程介绍

图 9-23　“添加新课程”界面

2) 测试代码

(1) 新建测试用例，选中 setUp( )和 tearDown( )复选框，之后出现添加课程的测试用例模板，如图 9-24 所示。

我们需要按测试用例模板来编写测试用例。

```java
package com.jwy.unit.test;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.jwy.dao.ICourseDao;
import com.jwy.dto.Course;

public class CourseTest extends BaseTest {

    private static ApplicationContext context;

    private static ICourseDao courseDao;

    @BeforeClass
    public static void setUpBeforeClass() {
        System.out.println("=====@BeforeClass=======");
        context = new ClassPathXmlApplicationContext("/applicationContext.xml");
        courseDao = (ICourseDao) context.getBean("courseDao");
    }

    @AfterClass
    public static void tearDownAfterClass() {
        System.out.println("=====@AfterClass=======");
        courseDao = null;
        context = null;
    }
```

图 9-24　填加新课程的测试用例模板

```
    @Before
    public void setUp() {
        System.out.println("=======@before=======");

    }

    @After
    public void tearDown() {
        System.out.println("=======@after=======");
    }

    @Test
    public void addCourse() {
        Course course = new Course();
        course.setName("artificial intelligence");
        course.setSchooltime("每周五上午8点到12点");
        course.setAddr("文科楼第三教室");
        course.setCredit(Short.valueOf("10"));
        course.setCourseInfo("通识课");
        course.setTeacherName("Eric");
        course.setTeacherInfo("一位优秀的教师");
        course.setIsFinish(false);
        course.setSpecialtyId(1);
        courseDao.insert(course);
    }

    @Test
    public void queryCourse() {
        String name = courseDao.findByID(1).getName();
        Assert.assertEquals("大学英语1", name);
    }
```

图 9-24(续)

(2) 根据“添加新课程”界面，这里编写了一个测试用例，如表 9-1 所示。

表 9-1　添加新课程，用户逐一将课程信息添加至文本框内

输入值	输入内容	预 期 输 出	实 际 情 况
课程名称：course.setName	artificialintelligence	artificialintelligence	与预期相同
上课时间：course.setSchooltime	每周五上午 8 点到 12 点	每周五上午 8 点到 12 点	与预期相同
上课地点：course.setAddr	文科楼第三教室	文科楼第三教室	与预期相同
课程学分：course.setCredit(Short.value0f)	10	10	与预期相同
课程介绍：course.setCourseInfo	通识课	通识课	与预期相同
授课教师：course.setTeacherName	Eric	Eric	与预期相同
教师介绍：course.setTeacherInfo	一位优秀的教师	一位优秀的教师	与预期相同
选课完成：course.setIsFinish	false	清空输入内容	与预期相同
选择专业：course.setSpecialtyId	1	跳转专业 1 状态	与预期相同
插入：courseDao.insert	course	插入成功	与预期相同

3) 测试顺序

JUnit 会以如下顺序执行测试(大致的代码)：

```
try {
    CalculatorTest test = new CalculatorTest ();   // 建立测试类实例
    test.setUp();                                   // 初始化测试环境
    test.testAdd();                                 // 测试某个方法
    test.tearDown();                                // 清理资源
}catch...
```

setUp( )用于建立测试环境，这里创建addCourse类的一个实例；tearDown( )用于清理资

源，如释放打开的文件等。以test开头的方法被认为是测试方法，JUnit会依次执行test×××( )方法。在testAdd( )方法中，对于add( )的测试选择两个数——10 和 50。如果方法的返回值与期待结果相同，assertEquals便不会产生异常。

如果有多个test×××( )方法，JUnit将会创建多个×××Test实例。每次运行一个test×××( )方法时，setUp( )和tearDown( )便会在test×××前后被调用。因此，不要在testA( )中依赖testB( )。

4) 测试结果

直接执行 Run|Run As|JUnit Test，就可以看到 JUnit 测试结果，如图 9-25 所示。

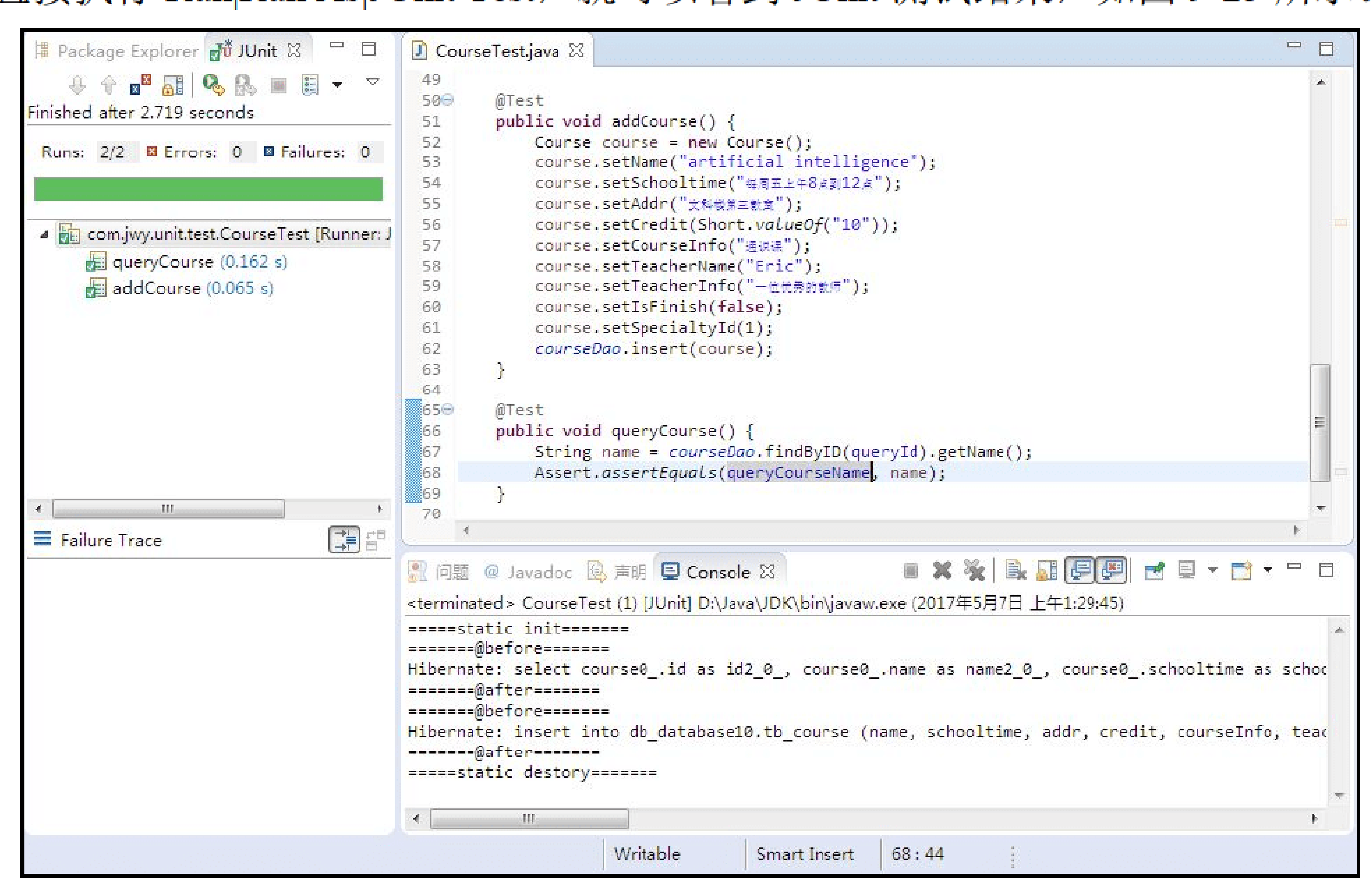


图 9-25　“添加新课程”测试成功界面显示

绿色表示测试通过，只要有一个测试未通过，就会显示红色并列出未通过测试的方法。可以试图改变 Beer 输入数据，然后运行 JUnit 就会报告错误。

下面简单总结一下上面用到的静态类 junit.framework.Assert 的使用方法，灵活地运用这些方法对用好 JUnit 进行测试是非常有帮助的。

(1) assertEquals( )方法：用来查看对象中存储的值是否是期待的值，与用于字符串比较的 equals( )方法类似。

(2) assertFalse( )和assertTrue( )方法：用来查看变量是否为false或true。如果assertFalse( )查看的变量的值是false，则测试成功；如果是true，则测试失败。assertTrue( )与之相反。

(3) assertSame( )和 assertNotSame( )方法：用来比较两个对象的引用是否相等和不相等，类似于通过= =和!=比较两个对象。

(4) assertNull( )和 assertNotNull( )方法：用来查看对象是否为空和不为空。

(5) fail( )方法：意为失败，用来引发错误。该方法有两个用途：一是在测试驱动开发中，由于测试用例都是在被测试的类之前编写，而写成时又不清楚其正确与否，此时就可以使用fail( )方法引发错误进行模拟；二是引发意外的错误，比如测试的内容是从数据库中读取的数据，要测试它们是否正确，而导致错误的原因却是数据库连接失败。

9.3.2　JUnit 下的覆盖测试工具 EclEmma

现在 IT 开发人员比以往任何时候都更加关注测试的重要性，没有经过良好测试的代码更容易出问题。在极限编程中，测试驱动开发已经被证明是一种能有效提高软件质量的方法。在测试驱动的开发方式中，软件工程师在编写功能代码之前首先要编写测试代码，这样便能从最开始保证程序代码的正确性，并且能够在程序每次演进时都进行自动的回归测试。测试对于软件产品的成败起着至关重要的作用，在极限编程领域，甚至有人提议任何未经测试的代码都应该自动从发布的产品中删除。尽管这种说法太过极端，但是测试本身的质量确实是一个需要高度关注的问题。测试的覆盖率是测试质量的一个重要指标，我们需要通过工具来帮助对软件测试覆盖结果进行考查。

EclEmma就是这样一个能帮助开发人员进行覆盖测试的优秀的Eclipse开源插件。EclEmma在覆盖测试领域是非常受欢迎的，它曾在 2006 年入围Eclipse Community Awards Winners决赛。虽然最后失败，但由此也可以看出 EclEmma 对开发人员确实能够提供很大的帮助。

1. EclEmma 介绍

提到 EclEmma，首先就要说到著名的 Java 覆盖测试工具——Emma。Emma 是一个在 SourceForge 上进行的开源项目。从某种程度上说，EclEmma 是 Emma 的一个图形界面。

Emma 的作者在开发 Emma 之初，程序员就已经有了各种各样优秀的开源 Java 开发工具。举例来说，有优秀的集成开发环境 Eclipse，有开源的 JDK，有单元测试工具 JUnit，有 Ant 这样的项目管理工具，还可以用 CVS 或 SubVersion 来进行源代码版本的维护。当时看来，也许唯一缺少的就是一个开源的覆盖测试工具了。Emma 就是为了填补这项空白而生的。现在的情况已经和 Emma 诞生的时候不一样了。时至今日，已经有了不少的覆盖测试工具。例如，Coverlipse 是一个基于 Eclipse 的覆盖测试插件，其他的还有 Cobertura、Quilt 和 JCoverage 等。但是，Emma 因具有一些非常优秀的特性而使得它更适合被广泛使用。和 Coverlipse 等工具比起来，Emma 是开源的，同时它对应用程序执行速度的影响非常小。

EclEmma 的出现弥补了 Emma 用户一个大的遗憾——缺乏图形界面以及对集成开发环境的支持。将 Eclipse 和 Emma 这两个在各自领域最为优秀的工具结合起来，这就是 EclEmma 所能提供的。

总之，EclEmma 是一个基于 Emma 的、免费的 Java 代码覆盖工具。它的目的是让用户可以在 Eclipse 工作平台上使用强大的 Java 代码覆盖工具 Emma。EclEmma 是非侵入式的，不需要修改项目或执行其他任何安装，它能够在工作平台中启动，并像运行 JUnit 测试一样直接对代码覆盖进行分析。覆盖结果将立即被汇总并在 Java 源代码编辑器中高亮显示。EclEmma 具有如下特点：快速的开发和测试周期，非常丰富的覆盖信息分析以及非入侵的测试方式。

2. EclEmma 测试环境的建立

EclEmma 插件的安装和其他大部分 Eclipse 插件的安装过程相同，既可以通过 Eclipse 标准的 Update 机制来远程安装 EclEmma 插件(如图 9-26 所示)，也可以通过从站点下载 ZIP

文件并解压到 Eclipse 所在的目录来安装。

可按如下步骤下载并直接安装 EclEmma：

(1) 从 http://sourceforge.net/projects/eclemma 下载 EclEmma 的安装压缩包(当前最新版本为 EclEmma 2.2.10)。

(2) 对压缩包解压缩，可以分别看到 eclemma-2.2.10 这个文件夹中有名为 features 和 plugins 的文件夹。

(3) 找到 Eclipse 在计算机中的安装位置，打开 Eclipse 安装文件，将会看到里面也会有相应的 features 和 plugins 这两个文件夹。

(4) 将文件夹 eclemma-2.2.10 中的 features 和 plugins 文件夹中的内容复制到 Eclipse 安装文件中相应的 features 和 plugins 文件夹中。

(5) 关闭 Eclipse 并重新启动。

(6) 在 Eclipse 界面中将会看到新增的“覆盖测试”按钮，如图 9-27 所示。

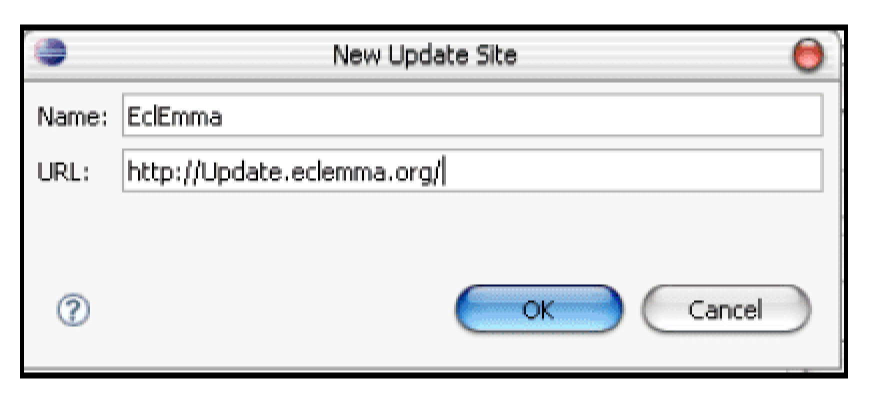


图 9-26　添加 EclEmma 更新站点

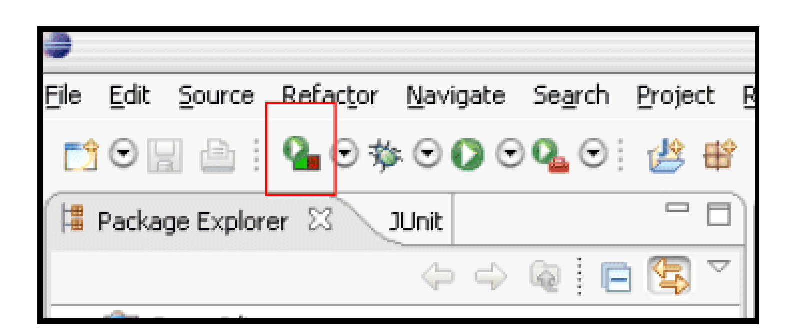


图 9-27　新增的“覆盖测试”按钮

3. EclEmma 的测试功能及使用流程

1) EclEmma 的测试功能

(1) 不同的颜色表示不同的测试情况。

在 Java 编辑器中，EclEmma 用不同的颜色标识源代码的测试情况。其中，绿色的行表示该行代码被完整执行，红色的行表示该行代码根本没有被执行，而黄色的行表明该行代码只有部分被执行。黄色的行通常出现在单行代码包含分支的情况下，如图 9-28 所示。

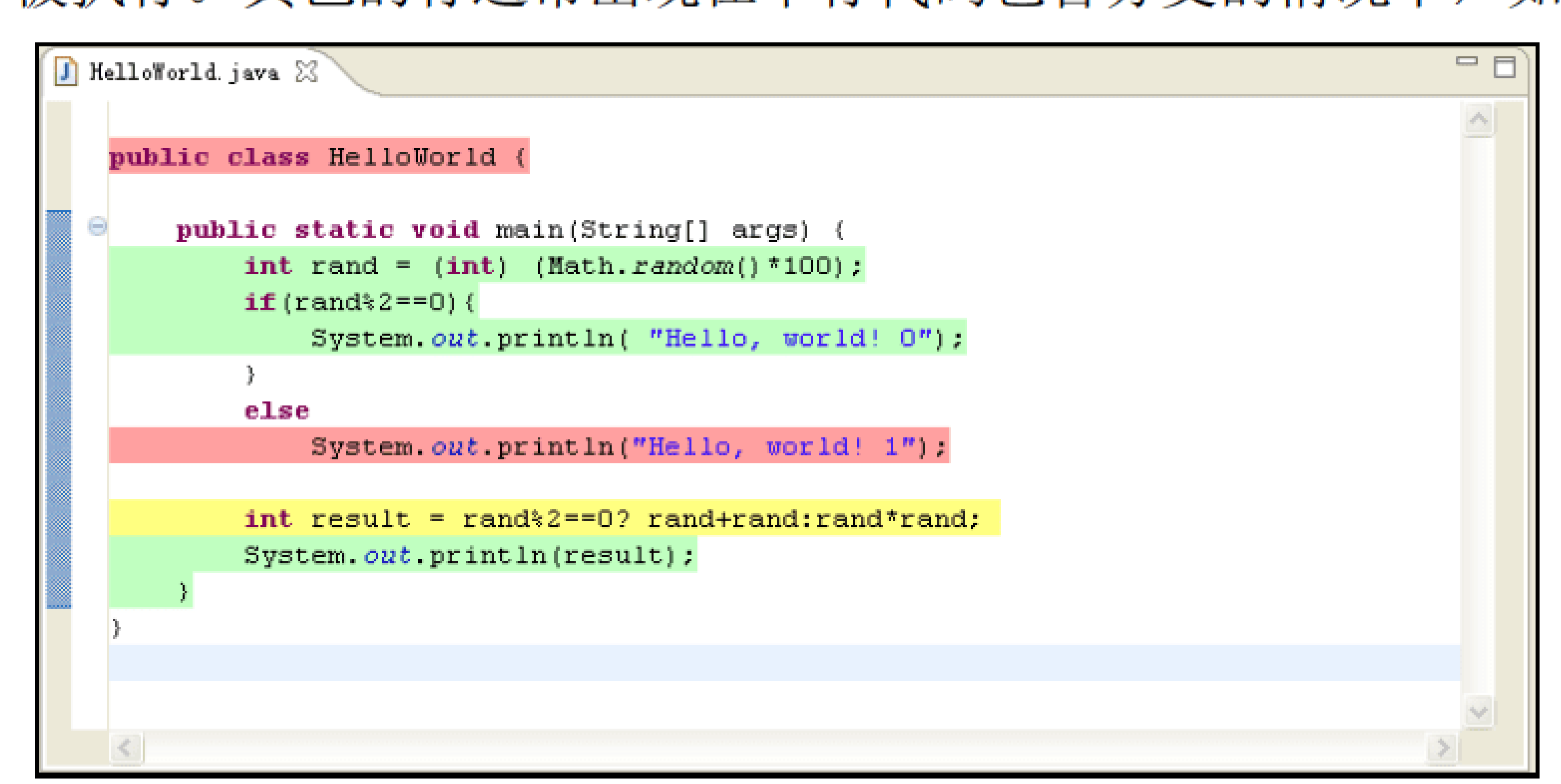
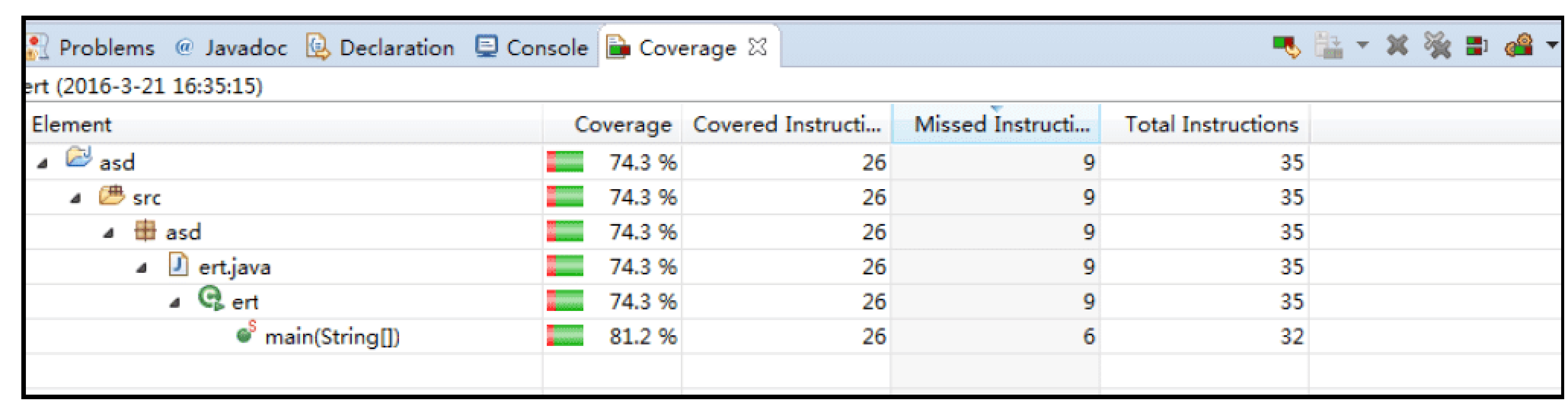


图 9-28　覆盖测试结果的显示

(2) 分层显示代码的覆盖测试率。

EclEmma 提供的 Coverage 视图能够分层显示代码的覆盖测试率。图 9-29 中的信息表明对 HelloWorld 的一次运行覆盖了 68.6%的代码。



Element	Coverage	Covered Instructi...	Missed Instructi...	Total Instructions
asd	74.3 %	26	9	35
src	74.3 %	26	9	35
asd	74.3 %	26	9	35
ert.java	74.3 %	26	9	35
ert	74.3 %	26	9	35
main(String[])	81.2 %	26	6	32

图 9-29　覆盖率统计

(3) 可以合并多次覆盖测试的结果。

想在一次运行中覆盖所有的代码通常会比较困难，如果能把多次测试的覆盖数据合并起来进行查看，那么就能更方便地掌握多次测试的测试效果。EclEmma 提供了这样的功能。

EclEmma 保存了所有的测试结果。通过 Coverage 视图的工具按钮就可以结合多次覆盖测试的结果，如图 9-30 所示。

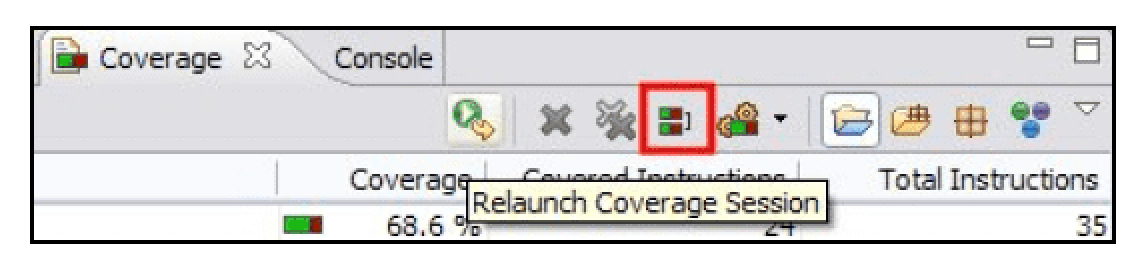


图 9-30　合并多次覆盖测试的结果

2) EclEmma 的使用流程

(1) 建立测试项目。为了实验 EclEmma 的特性，首先在 Eclipse 的 Workspace 中建立一个新的名为 test.Emma 的 Java 项目，如图 9-31 所示。

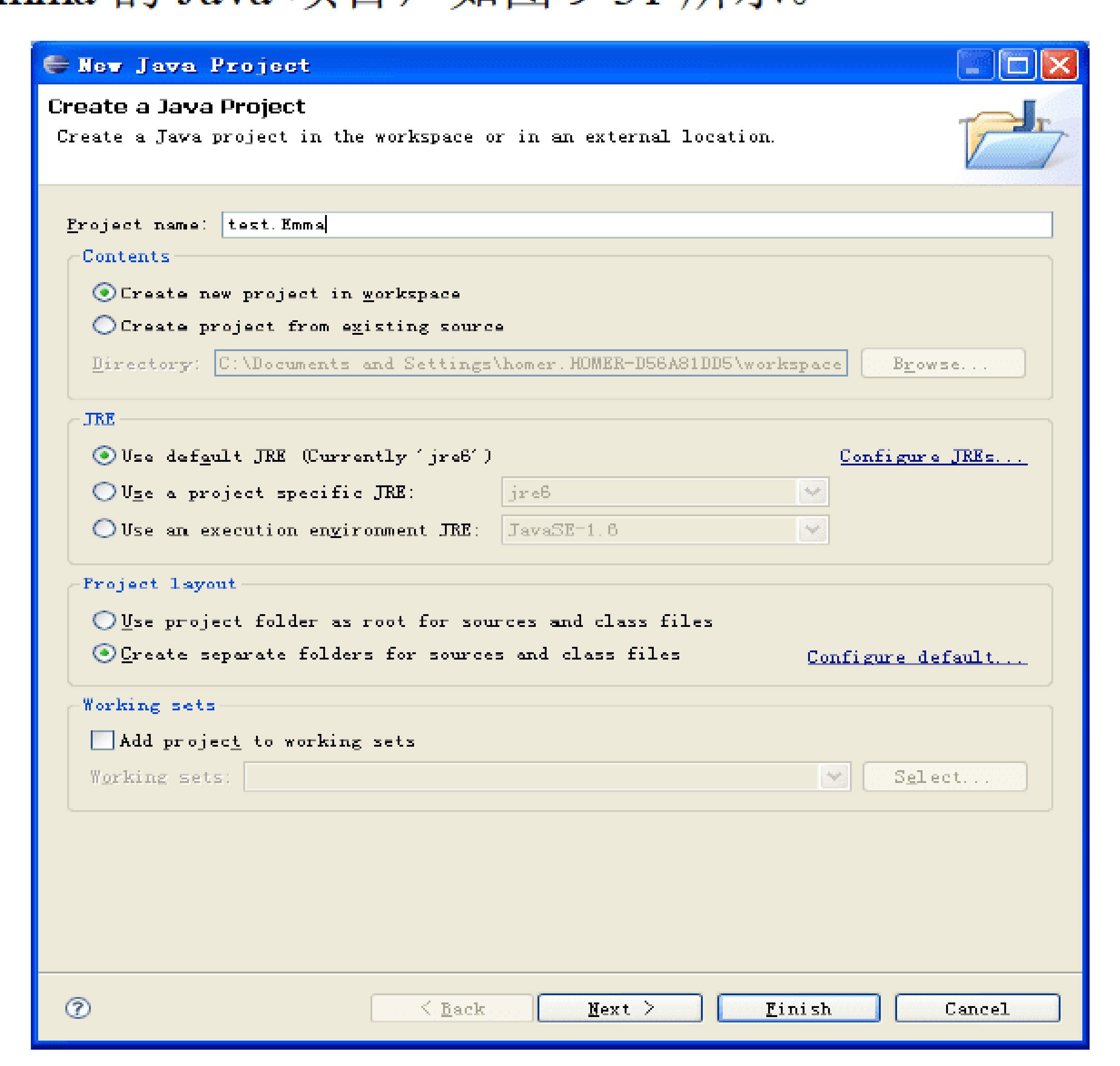


图 9-31　新建测试项目

(2) 编写用于测试 EclEmma 的代码，并在 test.Emma 这个 Java 项目的默认包中建立一个 HelloWorld 类：

```
package test.emma;
public class HelloWorld {
        /* @param args */
        public static void main(String[] args) {
```

```
                int rand = (int) (Math.random()*100);
                if(rand%2==0){
                        System.out.println( "Hello, world! 0");
                }
                else
                        System.out.println("Hello, world! 1");
                int result = rand%2==0? rand+rand:rand*rand;
                System.out.println(result);
        }
}
```

(3) 对 Java 应用程序进行覆盖测试，如图 9-32 所示。

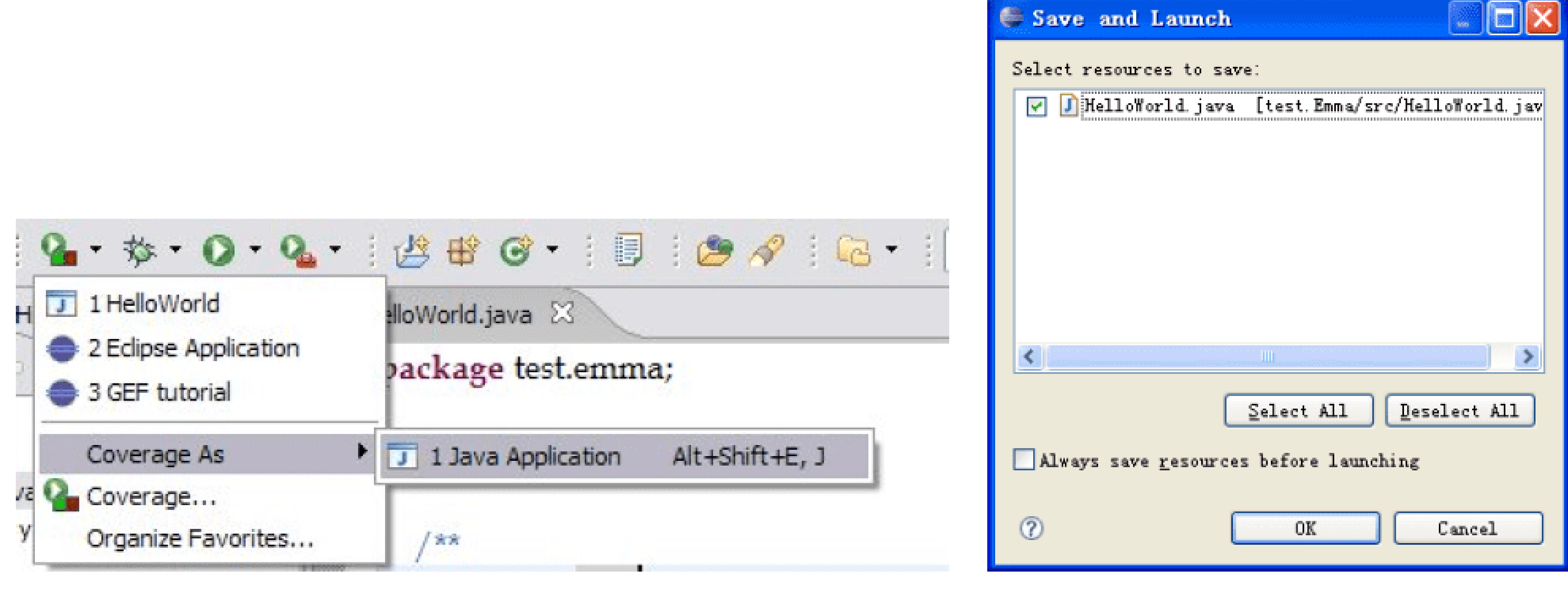


图 9-32　执行覆盖测试

覆盖测试执行完毕后，正在编辑 HelloWorld.java 的窗口将会变成前面图 9-28 所示的效果。

(4) 选择需要合并的覆盖测试结果，如图 9-33 所示。

(5) 查看合并后的覆盖测试结果，如图 9-34 所示。

由此可以看到，通过多次运行覆盖测试，最终代码达到了 91.4%的测试覆盖率。有趣的是，图 9-34 中第三行代码被标记为红色，而此行代码实际上是不可执行的。奥妙在于，没有生成任何 HelloWorld 类的实例，因此默认构造函数没有被调用，而 EclEmma 将这个特殊代码的覆盖状态标记在类声明的第一行。

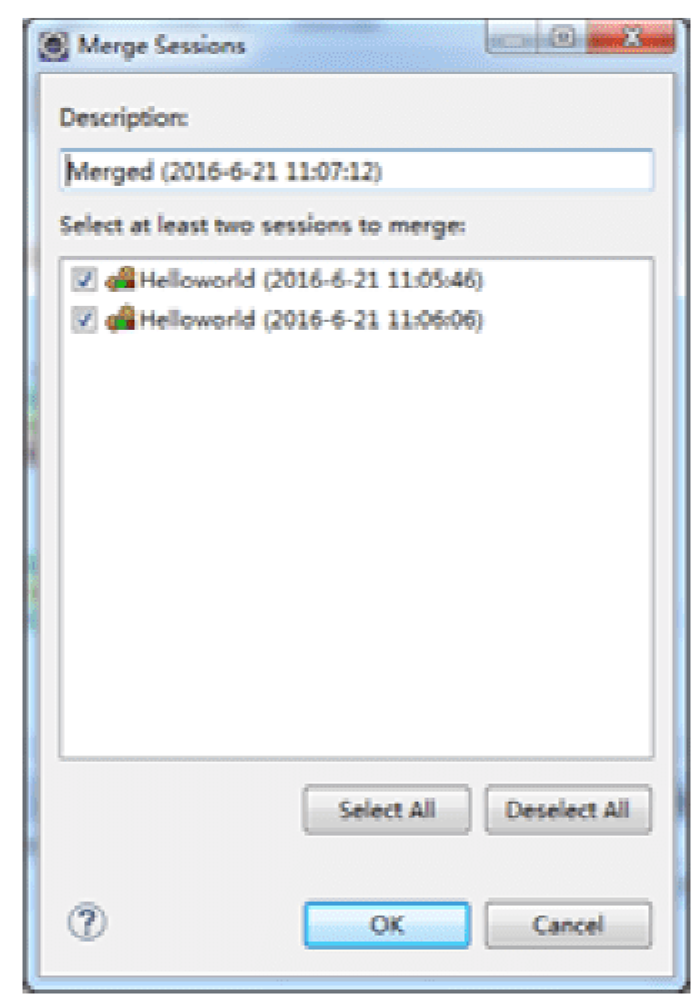


图 9-33　合并覆盖测试

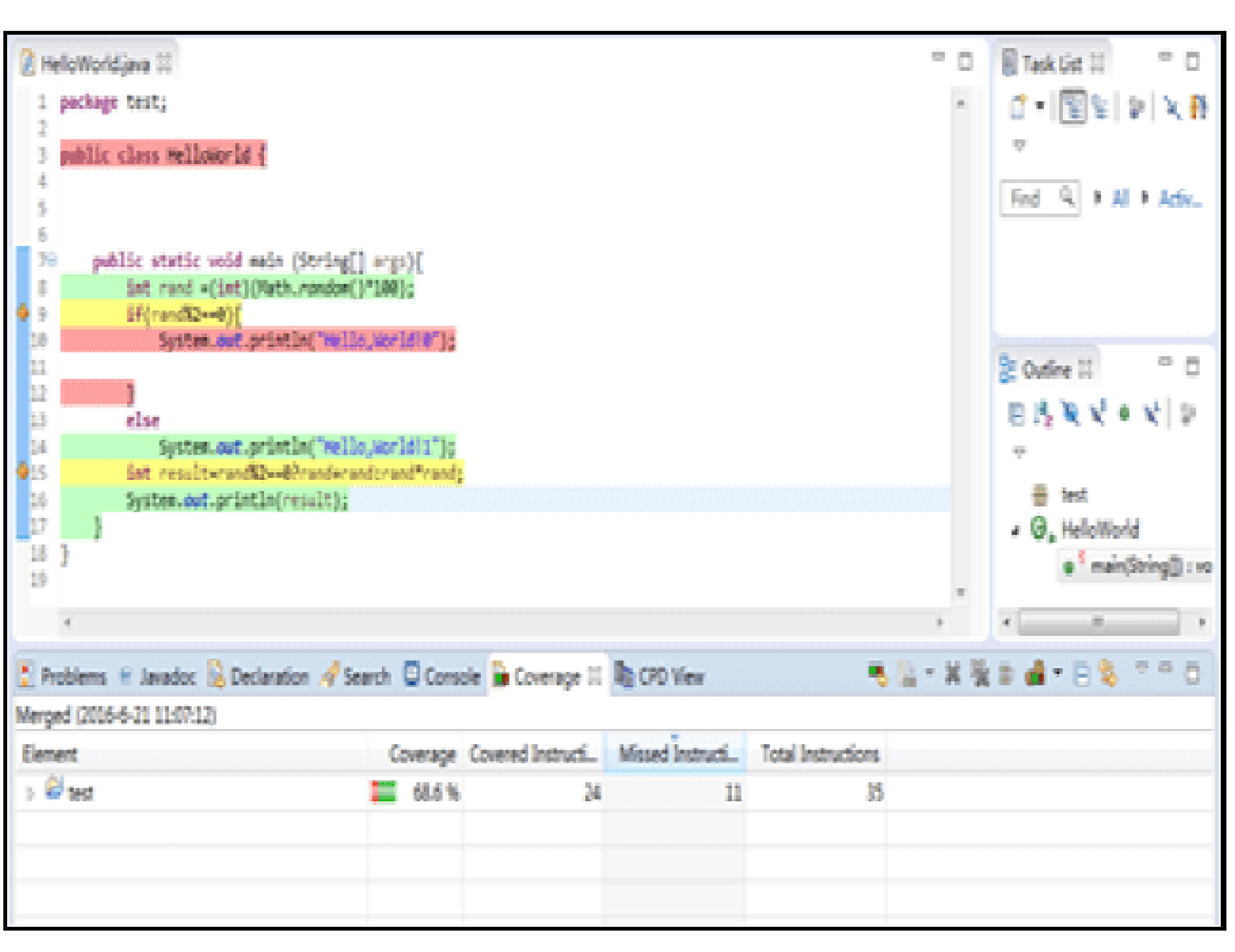


图 9-34　覆盖测试合并结果

3) EclEmma 的高级特性

如果 EclEmma 只能测试 Java 应用程序的测试覆盖率，那么相对命令行版本的 Emma 来说，它提供的增强功能就不多了。相反，EclEmma 提供了很多与 Eclipse 结合紧密的功能。它不仅能测试 Java 应用程序，还能计算 JUnit 单元测试以及对 Eclipse 插件测试的覆盖率。从图 9-35 中可以看出 EclEmma 目前支持 4 种类型的程序。

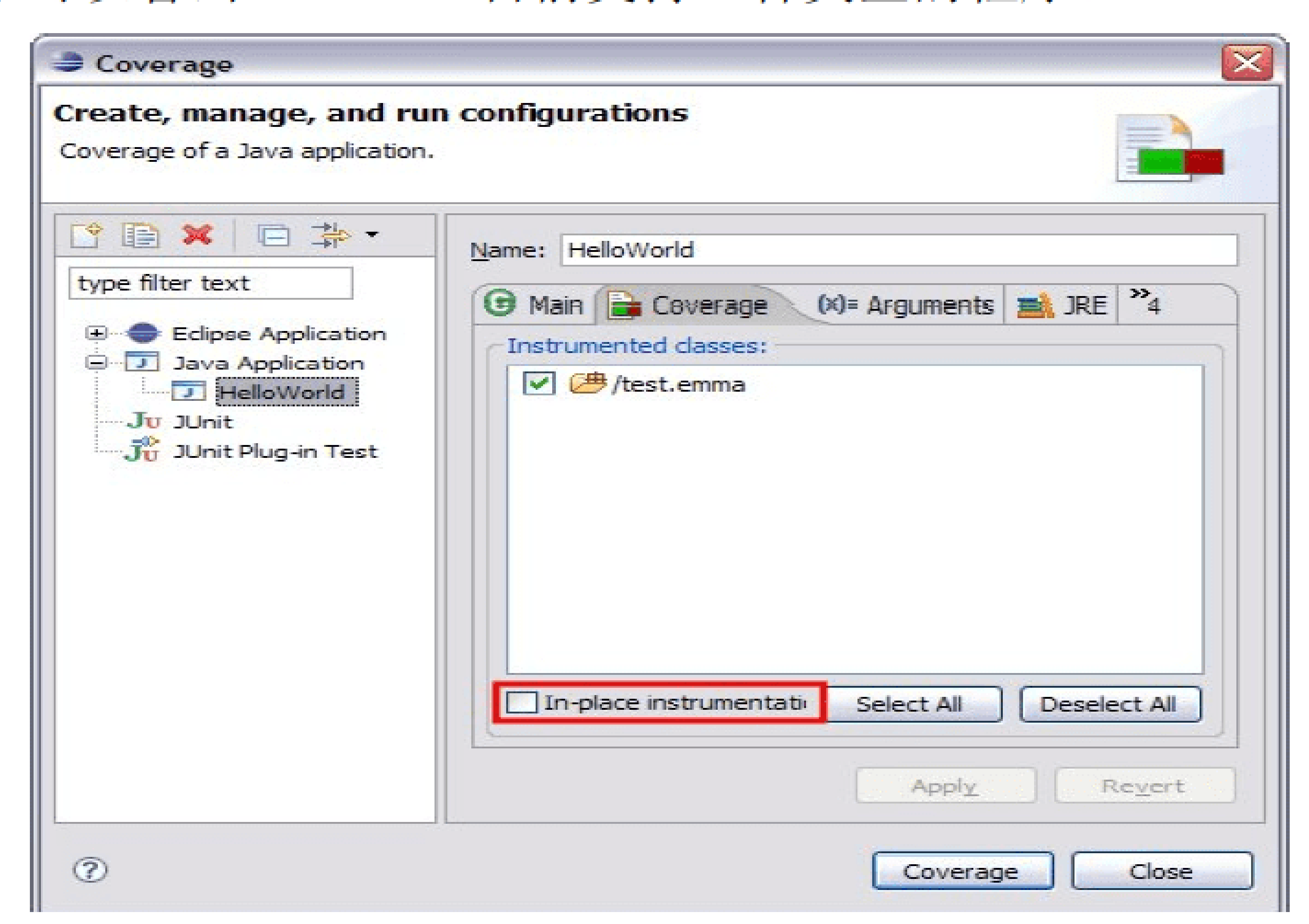


图 9-35　EclEmma 的配置界面

为了进一步了解 EclEmma 是如何获得覆盖测试数据的，需要先对 Emma 有个初步的了解。通常代码覆盖测试工具都需要对被执行的代码进行修改，Emma 提供了以下两种方式来完成这件事：

(1) 预插入模式：在对程序进行测试之前，需要用 Emma 中的工具对 class 文件或 jar 文件进行修改。修改后的代码可以立刻执行。覆盖测试的结果将会被存放到指定的文件中。

(2) 即时插入模式：即时插入模式不需要事先对代码进行修改。相反，对代码的修改是通过 Emma 定制的 class loader(类载入器)进行的。这种方式的优点很明显，不需要对 class 文件或 jar 文件进行任何修改。缺点是为了获得测试的结果，需要用 Emma 提供的命令 emmarun 来执行 Java 应用程序。

使用即时插入模式的优点很明显：class 文件和 jar 文件不会被修改。而预插入模式的应用范围更为广泛，对于某些需要嵌入到框架(例如 EJB)中运行的代码来说，只能使用预插入模式。EclEmma 仅使用 Emma 的预插入模式来工作，不过 EclEmma 默认会在临时目录中创建 class 文件和 jar 文件的副本来进行修改，因此在 workspace 中，class 和 jar 文件仍然保持原样。虽然听上去很好，但是由于需要修改 classpath 来使用修改过的 class 文件和 jar 文件，对于不能修改 classpath 的应用(例如 Eclipse RCP 和 JUnit Plugin Test)来说，还是只能选择修改 workspace 中的 class 文件和 jar 文件。对于 Java 应用程序和 JUnit 类型的覆盖测试来说，可以在配置对话框(见图 9-35)中选中 In-place instrumentation 复选框来指定直接修改 workspace 中的.class 文件和.jar 文件。

4. EclEmma 测试应用举例

下面仍以自动售货机程序为例进行覆盖测试。

这个程序的设计原理是这样的：顾客投入硬币来购买自己想要的饮料；已设定好两种

饮料——beer 和 orange；每种饮料的价格是 5 角钱；将两种饮料的数量设定为 3 个，5 角钱和 1 元钱的数量分别是 3 个。下面来看看自动售货机在不同状态下的行为：

(1) 如果顾客投入的正好是 5 角钱，那么可以选择自己想要的饮料，beer 或 orange。分别显示如下两条信息："You have pay for the beer. Please pick it up." 和 "You have pay for the orange. Please pick it up."。

(2) 如果选择的 beer 或 orange 的数量超过了设定的 3 个，那么自动售货机会报错，提示顾客饮料已售光，将显示信息："There has no beer，please pick up your money，Sorry!"。顾客将拿回自己的钱，得不到饮料。

(3) 如果顾客选择的是这两种饮料以外的其他饮料，那么系统将提示错误，因为只存在这两种饮料。

(4) 如果顾客投入的不是 5 角钱而是 1 元钱，那么顾客可以在饮料有剩余且有零钱找的情况下得到饮料及找回的零钱，显示如下信息："You have pay for the beer. Please pick it up and the loose change."或是"You have pay for the orange. Please pick it up and the loose change."。

(5) 如果顾客投入 1 元钱，但是饮料已经售完了，那么顾客将拿回自己的钱，得不到饮料。

(6) 如果顾客投入 1 元钱，饮料也有，但是没有零钱找给顾客，那么同样顾客将拿回自己的钱，得不到饮料。这时会显示："There has no loose change，Please pick up your money，Sorry!!!!!"。

(7) 如果顾客投入的既不是 5 角钱，也不是 1 元钱，那么自动售货机肯定无法找零钱，于是报错，显示信息："There has some input error!!!!!"。

1) 详细代码

```
public class SaleMachine {
    private int _beerNum，_orangeNum，_count_of_five，_count_of_one;
    private float _total;
    private String[] _type={"beer"，"orange"};
    private String _result;
    public SaleMachine()
    {
        init();
    }
    /*public SaleMachine(int five，int beer，int orange)      //便于测试的初始化函数
    {
        _count_of_one=one;
        _count_of_five=five;
        _beerNum=beer;
        _orangeNum=orange;
    }*/
    private void init()
    {
        _beerNum=3;
        _orangeNum=3;
        _count_of_five=3;
        _count_of_one=5;
    }
```

```
public String Operate(String type，int money)
//type 是用户选择的产品，money 是用户投币的种类
{
    float loose_change=0;
    if(money= =5)                       //如果用户投入 5 角钱
    {
        if(type.equals(_type[0]))       //如果用户选择啤酒
        {
            if(_beerNum>=1)             //如果还有啤酒
            {
                _beerNum– –;
                _count_of_five++;
                _result="You have pay for the beer. Please pick it up.";
                System.out.println(_beerNum);
                return _result;
            }
            else
                return "There has no beer，please pick up your money，Sorry!";
        }
        else if(type.equals(_type[1]))  //如果用户选择橙汁
        {
            if(_orangeNum>=1)
            {
                _orangeNum– –;
                _count_of_five++;
                _result="You have pay for the orange. Please pick it up.";
                System.out.println(_orangeNum);
                return _result;
            }
            else
                return "There has no orange，please pick up your money，Sorry!!";
        }
        else
            return "The type message is errno!!!";
    }
    else if(money= =1)                          //如果用户投入一元钱
    {
        if(type.equals(_type[0])&&_beerNum>=1)  //如果用户选择啤酒且还有啤酒
        {
            if(_count_of_five>=1)               //如果有零钱找
            {
                _beerNum– –;
                _count_of_five– –;
                _count_of_one++;
                _result="You have pay for the beer. Please pick it up and the loose change.";
                return _result;
            }
            else
                return "There has no loose change, Please pick up your money, Sorry!!!!";
        }
        else if(type.equals(_type[1])&&_orangeNum>=1)
        //如果用户选择橙汁且还有橙汁
```

```
                {
                    if(_count_of_five>=1)                //如果有零钱找
                    {
                        _orangeNum– –;
                        _count_of_five– –;
                        _count_of_one++;
                        _result="You have pay for the orange. Please pick it up and the loose change.";
                        return _result;
                    }
                    else
                        return "There has no loose change, Please pick up your money，Sorry!!!!!";
                }
            }
            return "There has some input error!!!!!!";
        }
    }
```

2) 用 Eclemma 进行覆盖测试

(1) 建立一个测试类，类名是 salemachineTest。在这个建好的工作界面上，可以看到如图 9-36 所示的几行代码，这是测试类在建立的过程中由软件自动生成的。

```
package SaleMachineTest;

import junit.framework.TestCase;

public class SaleMachineTest extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

}
```

图 9-36　生成的测试类

在最后两个大括号之间，就可以编写自己的测试程序了。

(2) 编写测试类的基本步骤。

① 扩展 TestCase 类。

② 覆盖 runTest( )方法(可选)。

③ 编写一些 test×××××( )方法。

根据上面的基本步骤编写每一个测试类，就这次测试的 Java 程序来看，大体的测试思路如下：

首先要保证顾客能够买到自己想要的饮料，也就是说，要把无论顾客投入5角钱还是1元钱，都能成功买到饮料的程序覆盖测试一遍。要保证这部分的覆盖测试，可以设计如下4个测试类：testSaleMachineA(顾客投入5角钱能够买到beer)、testSaleMachineB(顾客投入5角钱能够买到orange)、testSaleMachineC(顾客投入1元钱能够买到beer)、testSaleMachineD(顾客投入1元钱能够买到orange)。可以看到，此时的源程序界面视图变色，被测试覆盖的程序代码变成绿色，没有被测试覆盖的代码变成红色。如果有黄色的，则说明这些语句中没有完全被测试覆盖。图9-37所示是本次测试源程序变色的情况。

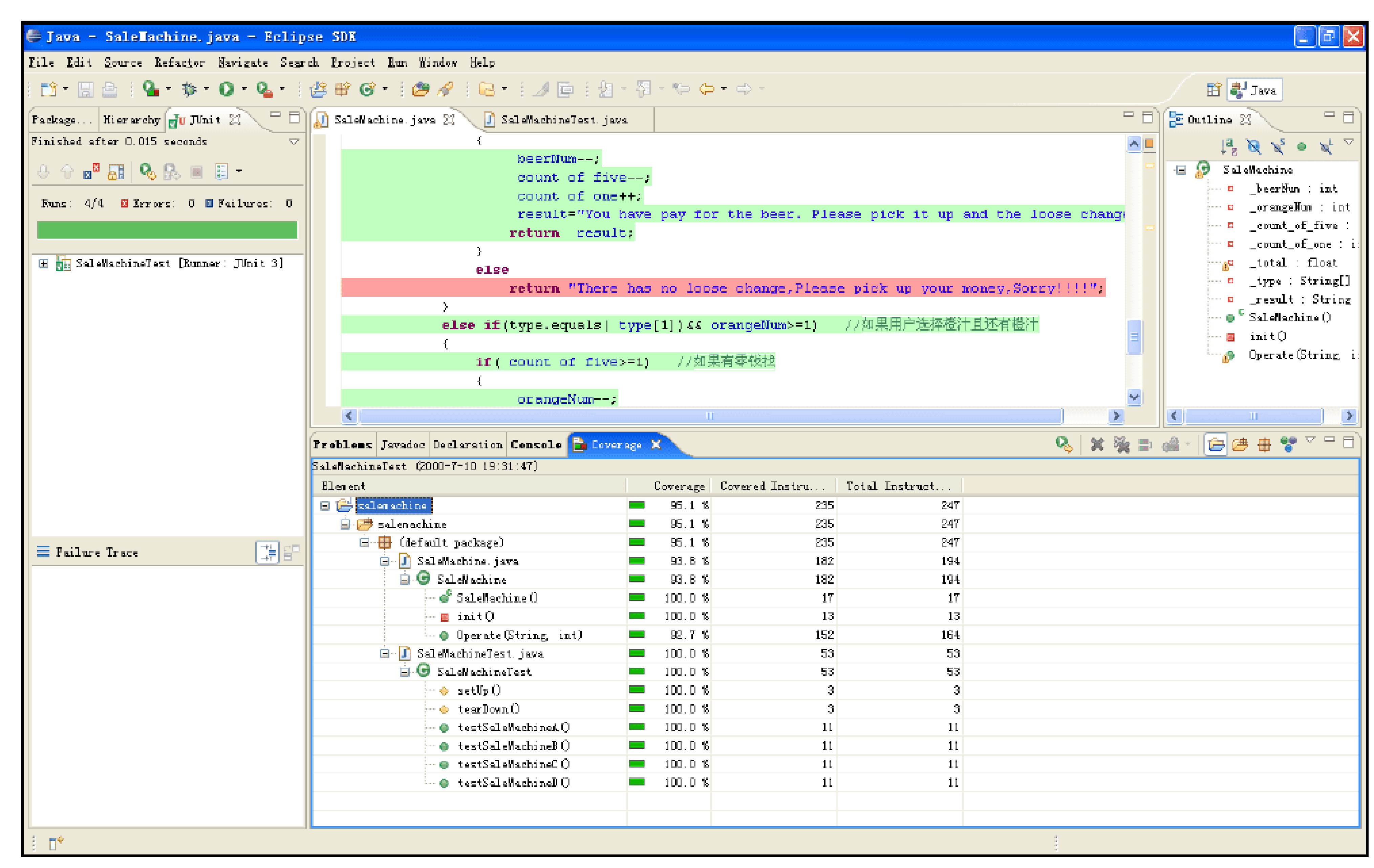


图 9-37　顾客能够买到饮料的覆盖测试结果

右下视图说明在这次覆盖测试中，测试覆盖率是 95.1%。

其次，如果成功执行了顾客可以用 5 角钱或 1 元钱买到 beer 或 orange 的情况，那么就要考虑顾客不能买到饮料的情况。这也要分两种情况来考虑：一种情况是顾客投入的正好是 5 角钱，不涉及找零钱的问题，这时只需要考虑两种饮料的数量，饮料卖完，后面的顾客将得不到饮料，并拿回自己的钱；另一种情况是顾客投入的是 1 元钱，这时不仅要考虑饮料是否还有，还要考虑是否有足够的零钱找给顾客，没有需要的饮料或没有零钱找给顾客都将导致顾客得不到饮料并拿回自己的钱。下面分别测试这两种情况。

① 顾客只投入 5 角钱。针对这种情况，就 beer 和 orange 分别设计了如下两个测试用例：testSaleMachineA_Error( )(顾客不能用 5 角钱买到 beer)，testSaleMachineB_Error( )(顾客不能用 5 角钱买到 orange)。鉴于 beer 和 orange 的数量被设为 3，于是我们的思路是：顾客前三次都可以买到自己需要的饮料，第四次则不能，系统提示顾客拿回自己的钱。

```
public void testSaleMachineA_Error() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("You have pay for the beer. Please pick it up.",
                salemachine.Operate("beer", 5));
        assertEquals("You have pay for the beer. Please pick it up.",
                salemachine.Operate("beer", 5));
        assertEquals("You have pay for the beer. Please pick it up.",
                salemachine.Operate("beer", 5));
        assertEquals("There has no beer，please pick up your money，Sorry!",
                salemachine.Operate("beer", 5));
}
```

同理，编写 testSaleMachineB_Error( )：

```
public void testSaleMachineB_Error() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("You have pay for the orange. Please pick it up.",
```

```
                    salemachine.Operate("orange", 5));
        assertEquals("You have pay for the orange. Please pick it up.",
                    salemachine.Operate("orange", 5));
        assertEquals("You have pay for the orange. Please pick it up.",
                    salemachine.Operate("orange", 5));
        assertEquals("There has no orange，please pick up your money，Sorry!!",
                    salemachine.Operate("orange", 5));
}
```

这时的测试结果如图 9-38 所示。

从图 9-38 可以看出，已经设计的 6 个测试类全部通过，且全部覆盖，从右下视图还可以看出测试覆盖率提高到了 97.4%。

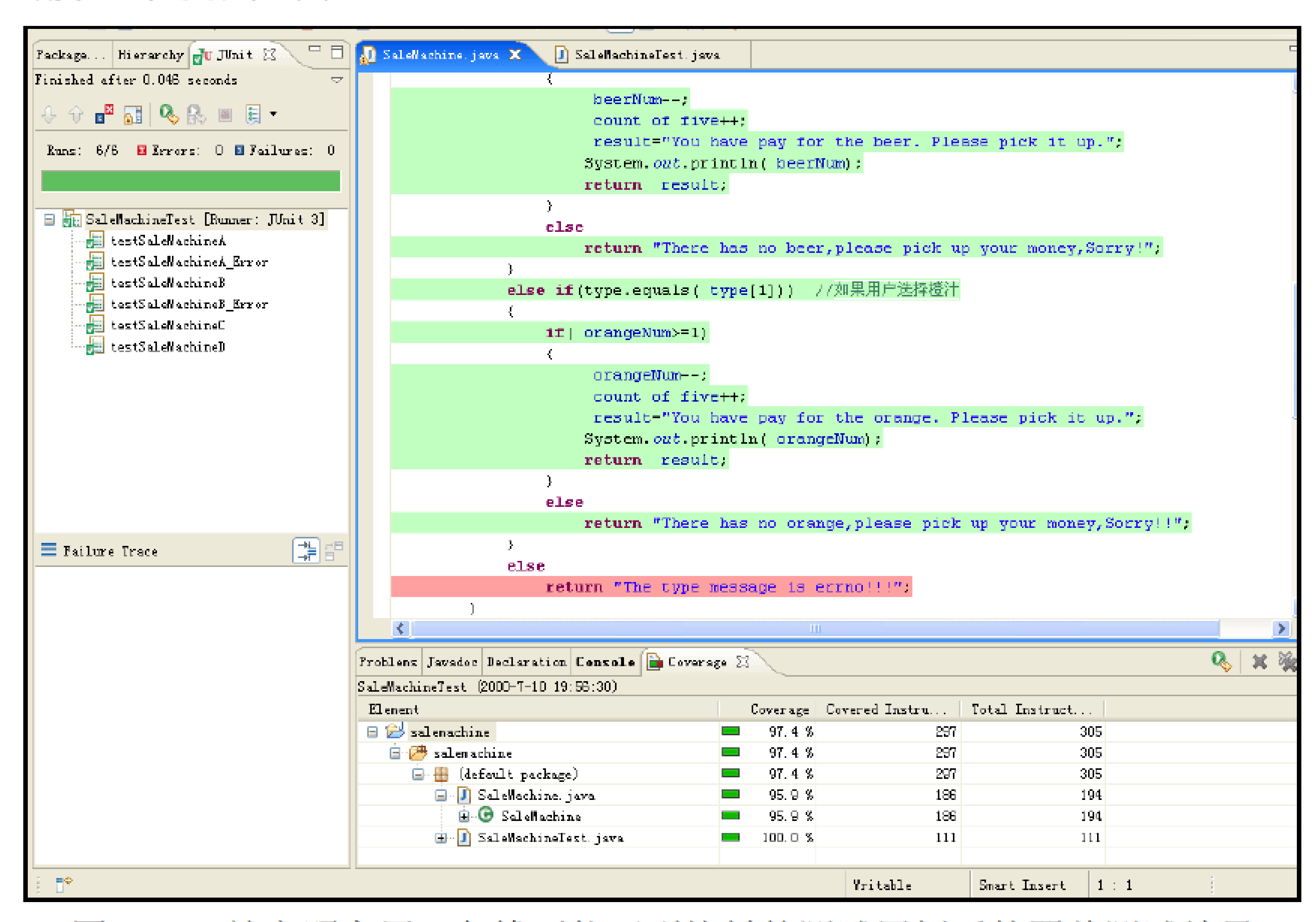


图 9-38　补充顾客用 5 角钱不能买到饮料的测试用例后的覆盖测试结果

图 9-39 显示了还需补充的测试用例警告。

```
SaleMachine.java    SaleMachineTest.java
                else
                    return "The type message is errno!!!";
            }
            else if(money==1)    //如果用户投入一元钱
            {
                if(type.equals( type[0])&& beerNum>=1)   //如果用户选择啤酒且还有啤酒
                {
                    if( count of five>=1)    //如果有零钱找
                    {
                        beerNum--;
                        count of five--;
                        count of one++;
                        result="You have pay for the beer. Please pick it
                        return  result;
                    }
                    else
                        return "There has no loose change,Please pick up your money,Sorry!!!!";
                }
                else if(type.equals( type[1])&& orangeNum>=1)    //如果用户选择橙汁且还有橙汁
                {
                    if( count of five>=1)    //如果有零钱找
                    {
                        orangeNum--;
                        count of five--;
                        count of one++;
                        result="You have pay for the orange. Please pick it up and the loose cha
                        return  result;
                    }
                    else
                        return "There has no loose change,Please pick up your money,Sorry!!!!!";
                }
            }
            return "There has some input error!!!!!";
        }
```

还有红的部分需要增加测试用例来使得所有的部分都变绿！！

图 9-39　还需补充的测试用例警告

② 然后考虑顾客投入1元钱时的情况。这个时候要保证有饮料、有零钱可以找回的情况下顾客才能得到饮料。因此，有无饮料、有无零钱是必须考虑的两个因素，缺一不可。缺少任一条件，都将使顾客无法得到饮料。因此设计了 testSaleMachineC_ Error( )(顾客投了1元钱但是不能买到 orange)和 testSaleMachineD_Error( )(顾客投了1元钱但是不能买到beer)这两个测试用例。

```
public void testSaleMachineC_Error() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("You have pay for the orange. Please pick it up and the loose change.",
                salemachine.Operate("orange", 1));
        assertEquals("You have pay for the orange. Please pick it up and the loose change.",
                salemachine.Operate("orange", 1));
        assertEquals("You have pay for the orange. Please pick it up and the loose change.",
                salemachine.Operate("orange", 1));
        assertEquals("There has no loose change，Please pick up your money，Sorry!!!!",
                salemachine.Operate("beer", 1));
}
```

同理，设计顾客投入 1 元买不到 orange 的情况：

```
public void testSaleMachineD_Error() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("You have pay for the beer. Please pick it up and the loose change.",
                salemachine.Operate("beer", 1));
        assertEquals("You have pay for the beer. Please pick it up and the loose change.",
                salemachine.Operate("beer", 1));
        assertEquals("You have pay for the beer. Please pick it up and the loose change.",
                salemachine.Operate("beer", 1));
        assertEquals("There has no loose change，Please pick up your money，Sorry!!!!!",
                salemachine.Operate("orange", 1));
}
```

将上述两个测试类加入测试程序，重新进行覆盖测试，可以得到如图 9-40 所示的源程序颜色变化情况。

由图 9-40 可以看到，不仅单元测试通过了，而且覆盖测试也顺利通过了，覆盖率达到了 98.9%。但仍有没有覆盖到的地方，我们要达到的覆盖率目标是 100%，这就说明还需要继续设计测试用例，以便覆盖测试能够真正做到整个 Java 程序的所有语句和分支都被覆盖测试一遍。

③ 来看一下仅剩的两条红色语句，那就是当顾客放进钱便直接报错，也就是说，顾客提出的要求自动售货机根本没有实现的可能。一种情况是：顾客放进的是5角钱或1元钱，但是他想喝别的饮料，这是不可能的，因为自动售货机里根本没有这种饮料。另一种情况是：顾客要的的确是beer或orange中的一种，但是却投进了100元，自动售货机根本没有找到足够零钱的可能，这时候也直接报错。针对这两种情况，可以设计testSaleMachineError( )(顾客投放5角钱或1元钱，但要喝的却是别的饮料)和testSaleMachine_FinalError( )(顾客要的是beer或orange，但是投了5角钱和1元钱之外的其他面值的货币)这两个测试用例，以保证覆盖测试可以涉及最后剩下的两条红色语句。

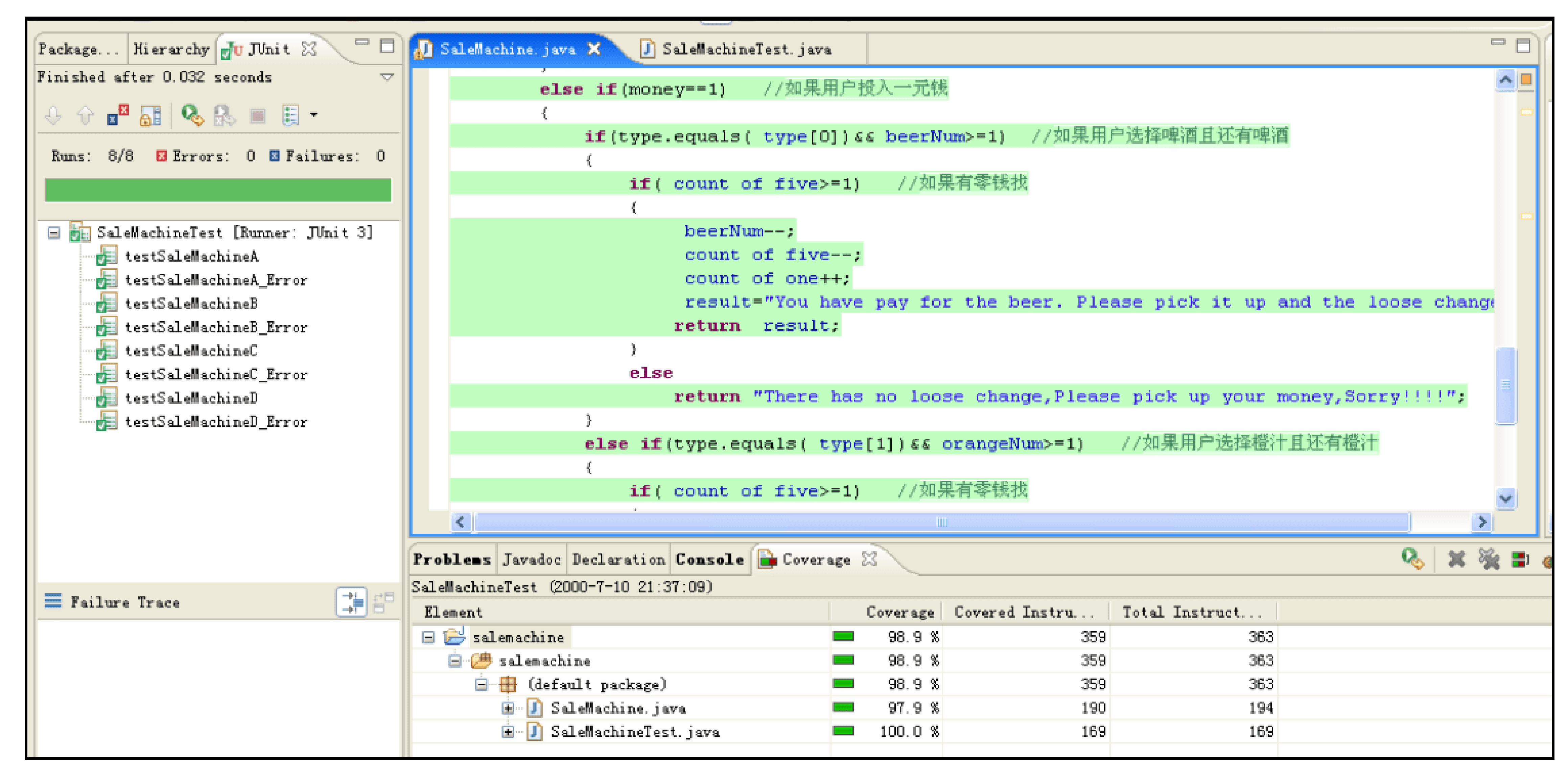


图 9-40　补充顾客用 1 元买不到饮料的测试用例后的覆盖测试结果

```
public void testSaleMachineError() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("The type message is errno!!!", salemachine.Operate("coca", 5));
    }
public void testSaleMachine_FinalError() {
        SaleMachine salemachine=new SaleMachine();
        assertEquals("There has some input error!!!!!", salemachine.Operate("beer", 100));
        assertEquals("There has some input error!!!!!", salemachine.Operate("orange", 100));
}
```

将这两个测试用例添加到测试程序中，看一下最后的测试结果：全变绿了，覆盖率已经达到 100%，如图 9-41 所示，这说明测试覆盖到了源程序中的所有语句。

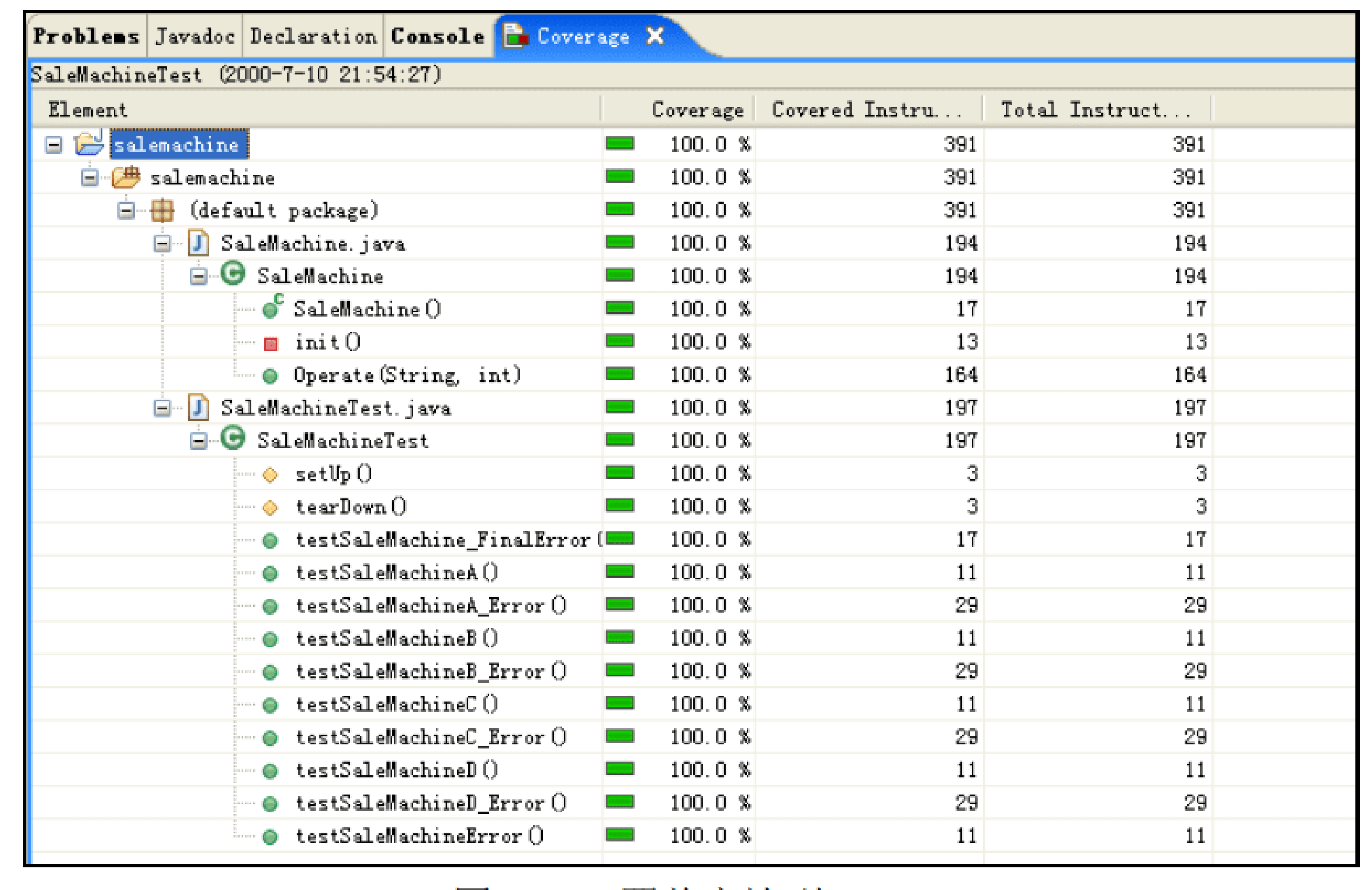


图 9-41　覆盖率达到 100%

用 EclEmma 仅能进行语句覆盖和分支覆盖等比较初级的覆盖测试，对于高级别的覆盖项目，就要用到商用白盒测试工具了。尽管如此，EclEmma 能够很好地体现覆盖测试的

思想，帮助读者掌握覆盖测试的知识和技能。

习题和思考题

1. 什么是单元测试？单元测试一般包括哪些内容？如何进行单元测试？
2. 单元测试关注的技术标准有哪些？作用是什么？
3. 搜索并下载其他类型的开源单元测试工具，与 JUnit 进行全面比较，并给出它们详细的应用流程。
4. 基于 JUnit 对已选课程信息统计等功能程序实施覆盖测试，设计出各种测试用例，完成 100%的覆盖测试。

第10章　软件集成测试和确认测试

10.1　集成测试

1999年美国火星气象卫星脱轨，用了5万美元进行问题查找，最后发现控制软件中有两个模块使用了不同的加速度单位。此例说明，通过单元测试的模块，还需要进行集成测试。

集成测试的重要性是其测试需求所固有的，研究表明，集成错误比单元错误的修改成本要昂贵 30 倍，在开发后期发现的错误，其中大部分是集成错误。

10.1.1　集成测试的概念

集成测试又叫组装测试或联合测试，是单元测试的多级扩展，是在单元测试的基础上进行的一种有序测试。这种测试需要将所有模块按照设计要求，逐步装配成高层的功能模块，并进行测试，直到整个软件成为一个整体。集成测试的目的是检验软件单元之间的接口关系，以期望通过测试发现各软件单元接口之间存在的问题，最终把经过测试的单元组成符合设计要求的软件。集成测试验证程序和概要设计说明的一致性，任何不符合该说明的程序模块行为都应该加以记载并上报。因此，集成测试是发现和改正模块接口错误的重要阶段。

1. 为什么要开展集成测试

通常单元测试后的各个程序单元都可以正常地工作，为什么还要把它们组装在一起进行测试，看它们是否能正常工作呢？这是因为在将单元组装成一个整体时，我们需要考虑相关问题：①在把各个单元模块连接起来的时候，穿越模块接口的数据是否会丢失；②一个单元模块的功能是否会对另一个单元模块的功能产生不利的影响；③各个子功能组合起来，能否达到预期要求的父功能；④全局数据结构是否有问题；⑤共享资源访问是否有问题；⑥单个模块的误差积累起来，是否会放大，从而达到不能接受的程度；⑦引入一个模块后，是否会对其他与之相关的模块产生负面影响。

集成测试有以下不可替代的特点：

(1) 单元测试具有不彻底性，对于模块间接口信息内容的正确性、相互调用关系是否符合设计无能为力。只能靠集成测试来进行保障。

(2) 同系统测试相比，由于集成测试用例是从程序结构出发的，目的性、针对性更强，因此测试发现问题的效率更高，定位问题的效率也较高。

(3) 能够较容易地测试到系统测试用例难以模拟的特殊异常流程，从纯理论的角度来讲，集成测试能够模拟所有实际情况。

(4) 定位问题较快，由于集成测试具有可重复性强、对测试人员透明的特点，发现问题后容易定位，因此能够有效地加快进度，减少隐患。

集成测试在软件分级测试中具有很重要的意义，具体体现在：

(1) 在单元测试和系统测试间起到承上启下的作用，既能发现大量单元测试阶段不易发现的接口类错误，又可以保证在进入系统测试前及早发现错误，减少损失(事实上，对于系统而言，接口错误是最常见的错误)。

(2) 单元测试通常是单人执行，而集成测试通常是多人执行或第三方执行。集成测试通过模块间的交互作用和不同人的理解及交流，更容易发现实现上、理解上的不一致和差错。

表 10-1 给出了单元测试、集成测试与系统测试的差别。

表 10-1　单元测试、集成测试与系统测试的差别

测试类型	对象	目的	测试依据	测试角度	测试方法
单元测试	模块内部的程序错误	消除局部模块的逻辑以及功能上的错误和缺陷	模块逻辑设计、模块外部说明	站在开发人员角度	大量采用白盒测试方法
集成测试	模块间的集成和调用关系	找出与软件设计相关的程序结构、模块调用关系、模块间接口方面的问题	程序结构设计、软件概要设计说明书	站在测试人员角度	灰盒测试，采用较多黑盒方法构造测试用例
系统测试	整个系统，包括系统中的硬件等	对整个系统进行一系列的整体、有效性测试	系统结构设计、目标说明书、需求说明书等	站在用户使用角度	黑盒测试

因此，在进行单元测试后，有必要进行集成测试，发现并排除在模块连接中可能发生的上述问题，最终构成要求的软件子系统或系统。

理论上，凡是两个单元(如函数单元)的组合测试都可以叫作集成测试。实际操作中，通常集成测试的对象为模块级的集成和子系统间的集成，其中子系统集成测试称为组件测试，如图 10-1 所示。

对于传统软件来说，按集成粒度不同，可以把集成测试分为 3 个层次，即模块内集成测试、子系统内集成测试以及子系统间集成测试。

对于面向对象应用系统来说，按集成粒度不同，可以把集成测试分为两个层次：类内集成测试和类间集成测试。

2. 集成测试的内容

集成测试的内容包括以下两个方面：

(1) 功能性测试主要解决前面提到的几个问题，侧重于测试软件模块组装以后的功能有没有达到预期效果，这是集成测试最主要的部分。

(2) 集成测试还应包含其他一些测试，包括资源冲突、任务优先级冲突、性能和稳定性在内的兼容性、可靠性、可用性、效率、可维护性等测试内容。

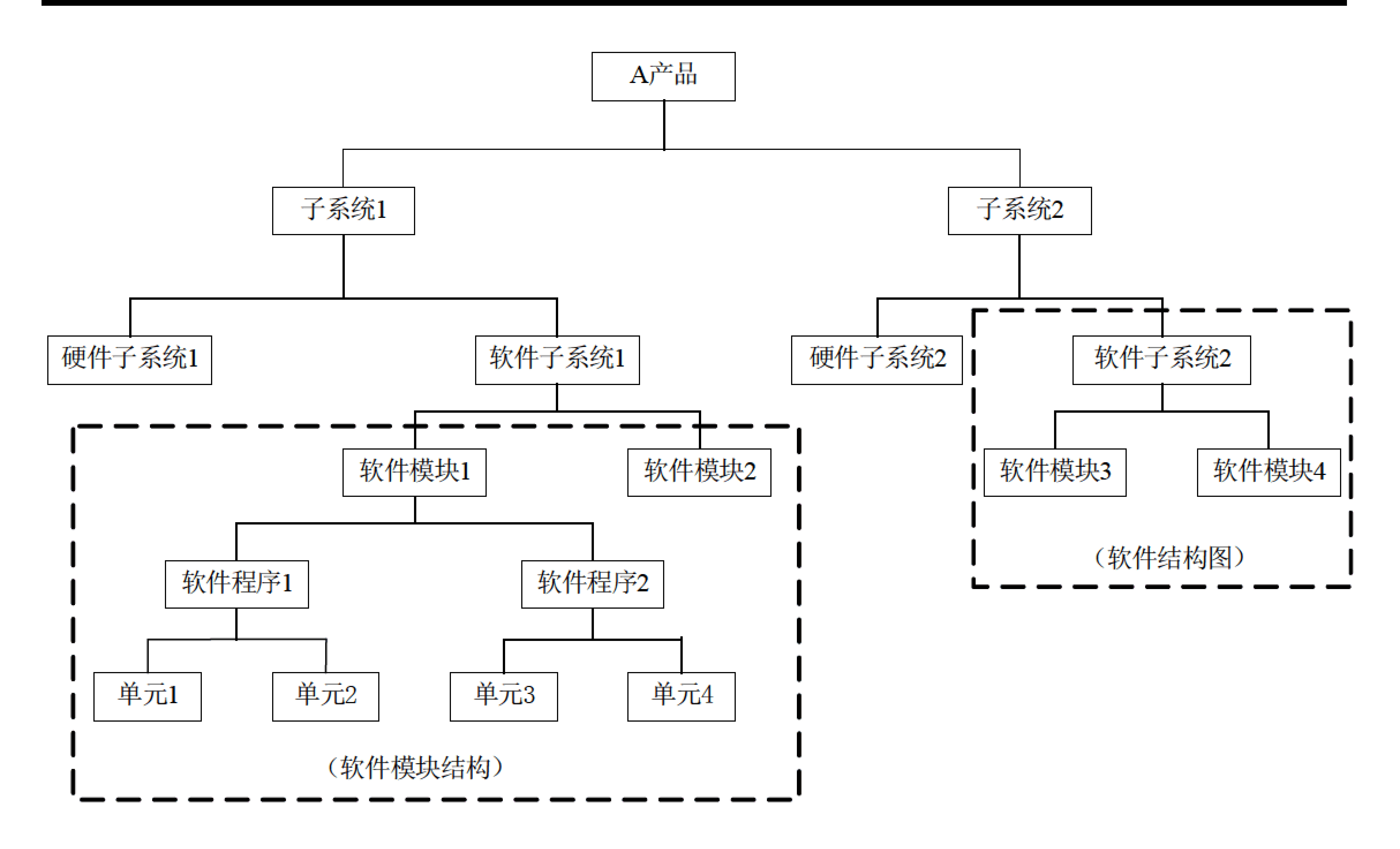


图 10-1　软件系统结构图

1) 集成后的功能性测试

基于集成单元实现的功能测试集成后的功能(合一)，考查多个模块间的协作，既要满足集成后要实现的复杂功能，也不能衍生出不需要的多余功能(错误功能)。此时需要关注：被测对象的各项功能是否实现；异常情况是否有相关的错误处理；模块间的协作是否高效合理。

2) 接口测试

模块间的接口包括函数接口和消息接口。对函数接口的测试，应关注函数接口参数的类型和个数的一致性、输入/输出属性的一致性、范围的一致性；对消息接口的测试，应关注收发双方对消息参数的定义是否一致、消息和消息队列长度是否满足设计要求、消息的完整性如何、消息的内存是否在发送过程中被非法释放、有无对消息队列阻塞进行处理等。

3) 全局数据结构测试

全局数据结构往往存在被非法修改的隐患，因此对全局数据结构的测试主要关注以下内容：全局数据结构的值在两次被访问的间隔是可预知的；全局数据结构的各个数据段的内存不应被错误释放；多个全局数据结构间是否存在缓存越界；多个软件单元对全局数据结构的访问应采用锁保护机制。

4) 资源测试

资源测试包括共享资源测试和资源极限测试。

共享资源测试常应用于数据库测试和支撑测试。共享资源测试需要关注：是否存在死锁现象；是否存在过度利用情况；是否存在对共享资源的破坏性操作；公共资源访问锁机制是否完善。

资源极限测试关注系统资源的极限使用情况以及软件对资源耗尽时的处理，保证软件

系统在资源耗尽的情况下不会出现系统崩溃。

5) 性能测试

依据某个部件的性能指标进行性能测试，及时发现性能瓶颈。

在多任务环境中，还需要测试任务优先级的合理性，需要考虑：实时性要求高的功能是否在高优先级任务中完成；任务优先级设计是否满足用户操作相应时间的要求。

6) 稳定性测试

稳定性测试要关注：是否存在内存泄漏而导致长期运行资源耗竭；长期运行后是否出现性能的明显下降；长期运行是否出现任务挂起等。

3. 集成测试的步骤

集成测试可按以下六步进行。

1) 体系结构分析

首先跟踪需求分析，对要实现的系统划分出结构层次图。

其次，是对系统各个组件之间的依赖关系进行分析，然后据此确定集成测试的粒度，即集成模块的大小。

2) 模块分析

一般可从这些角度进行模块分析：①确定本次要测试的模块；②找出与该模块相关的所有模块，并且按优先级对这些模块进行排列；③从优先级别最高的相关模块开始，把被测模块与其集成到一起；④依次集成其他模块。

3) 接口分析

接口的划分要以概要设计为基础，一般通过这些步骤来完成：①确定系统的边界、子系统的边界和模块的边界；②确定模块内部的接口；③确定子系统内模块间接口；④确定子系统间接口；⑤确定系统与操作系统的接口；⑥确定系统与硬件的接口；⑦确定系统与第三方软件的接口。

4) 风险分析

风险通常被分为 3 种类型：①项目风险(包括项目管理和项目环境的风险)；②商业风险(它和领域的相关概念及规则息息相关)；③技术风险(这是针对应用程序的具体实现而言的，主要和代码级的测试有关)。

风险分析是一个定义风险并且找出阻止潜在的问题变成现实的方法的过程。

通常把风险分析分为 3 个阶段：风险识别、风险评估和风险处理。

5) 可测试性分析

必须尽可能早地分析接口的可测试性，提前为后续的测试工作做好准备。

6) 集成测试策略分析

集成测试策略分析的主要任务就是根据被测对象选择合适的集成测试策略。

10.1.2　传统的集成测试方法

由模块组装成程序时，有两种方法。一种方法是先分别测试每个模块，再把所有模块按设计要求放在一起结合成想要的程序，这种方法称为非渐增式集成测试方法；另一种方法是把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来进行测试，这种每次增加一个模块的方法称为渐增式集成测试，这种方法实际上同时完成了单元测试和集成测试。

非递增式集成测试的优点是测试过程中基本不需要设计开发测试工具。不足是对于复杂系统，当出现问题时故障定位困难，和系统测试接近，难以体现和发挥集成测试的优势。

递增式集成测试逐渐集成，由小到大，边集成、边测试，测完一部分，再连接一部分。在复杂系统中，划分的软件单元较多，通常是不会一次集成的。

软件集成的精细度取决于集成策略。通常的做法是先模块间的集成，再部件间的集成。这样做的优点是测试层次清晰，出现问题能够快速定位。缺点是需要开发测试驱动和桩。

图 10-2 中的矩形表示程序中的单元，单元之间的连线表示程序的调用层次。单元 A 调用单元 B、C、D，单元 B 调用单元 E，单元 D 调用单元 F。

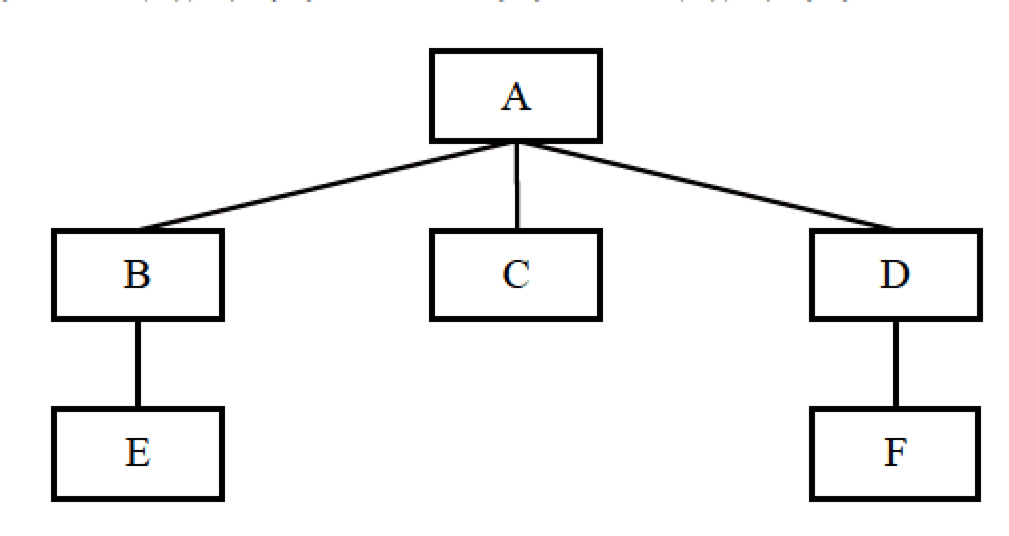


图 10-2　程序结构示意图

1. 一次性组装方式(或称大爆炸集成方法)

一次性组装方式是一种非渐增式集成测试方法，也可以叫作整体拼装。使用这种方式，首先对每个模块分别进行单元测试，单元的测试次序是无关紧要的，可以顺序地进行，也可以平行地进行。最后把通过单元测试的模块组装在一起进行测试，最终得到要求的软件系统。图 10-3 演示了这个过程。

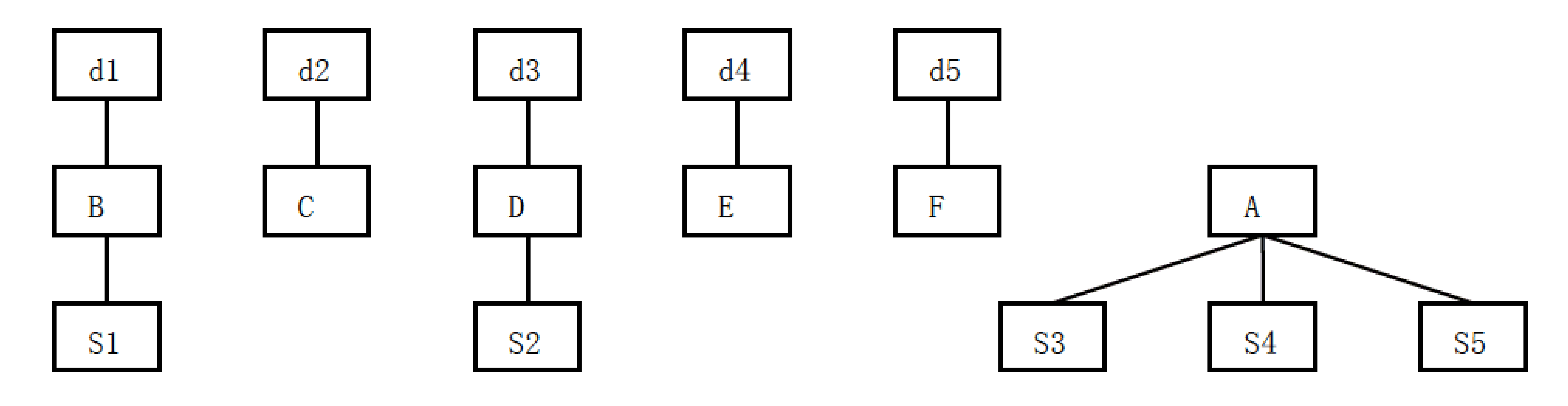


图 10-3　非渐增式集成测试

在图 10-3 中，模块 d1、d2、d3、d4、d5 是对各个模块做单元测试时间里的驱动模块，s1、s2、s3、s4、s5 是为单元测试而建立的桩模块。这种一次性组装方式试图在辅助模块的协助下，在分别完成模块单元测试的基础上，将所有测试模块连接起来进行测试。但是由于程序中不可避免地存在涉及模块间接口、全局数据结构等方面的问题，因此一次试运

行成功的可能性并不是很大。其结果通常是发现错误，但是不知道去哪里找原因。查错和改错都非常困难。

非渐增式集成测试方法有一个贬义的名称——“莽撞测试”，它的意思是一下子把几十个甚至上百个单元很莽撞地连接在一起。显然，使用这种方法的人希望最后的组装阶段会大大缩短，并且所有的单元大体上能在一起较好地执行，这对进度比较紧的项目非常具有诱惑力，但实际效果常常是相反的。用这种方法会产生一系列的问题，在我们讲述了渐增式集成测试方法后，通过对比就能看到这些问题。

2. 渐增式集成测试

渐增式集成测试方法不是独立地测试每个单元，而是首先把下一个要测试的单元同已经测试过的单元集合组装起来，然后再测试，在组装的过程中边连接边测试，以发现连接过程中产生的问题，最后通过渐增式方法逐步组装成要求的软件系统。以不同的组合方式可以有很多的渐增式集成测试方法。典型的有自顶向下和自底向上两种。

1) 自顶向下集成测试方法

自顶向下集成测试是按照程序和控制结构从主控模块开始，向下逐个把模块连接起来。把附属主控模块的子模块、孙模块等组装起来的方式有两种：深度优先和广度优先。

(1) 深度优先方法

深度优先方法(如图 10-4 所示)是先把结构中一条主要的控制路径上的全部模块组装起来。主要路径的选择与特定的软件应用特性有关，可以尽可能地选取程序主要功能所涉及的路径。

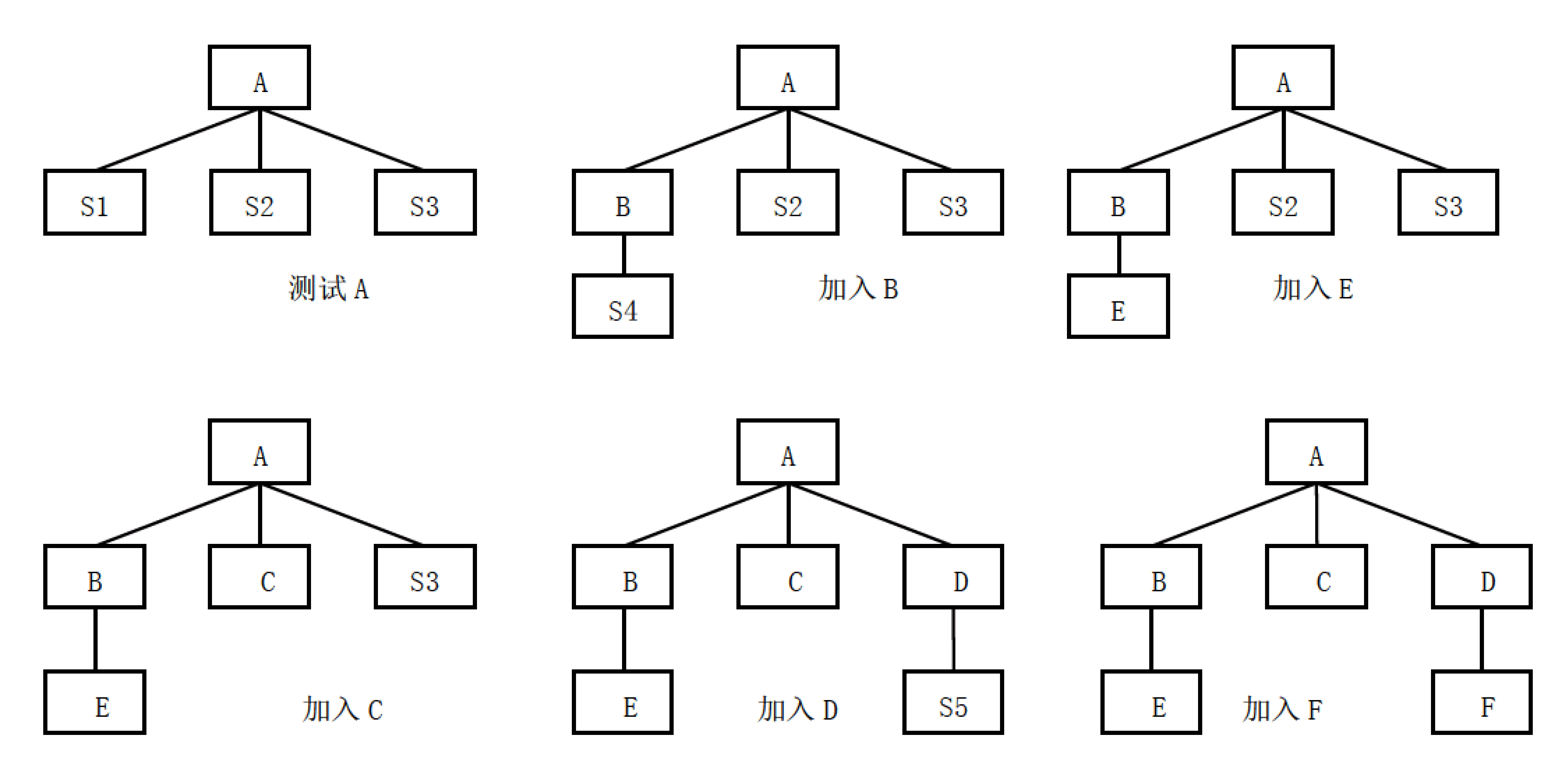


图 10-4　自顶向下按深度方向组装的例子

(2) 广度优先方法

从结构的顶层开始逐层向下组装。把上一层模块直接调用的模块组装进去，然后对于每一个新组装进去的模块，再把其直接调用的模块组装进去。参见图 10-5，从模块 A 出发，先组装模块 B、C、D，接着是模块 E、F 这一层，以此类推。

深度优先和广度优先自身也存在不同的组装次序，这些选择次序一般说来无所谓，但

必须遵循自顶向下的原则：新组装模块的上层必须被测试过。可以从以下两点来注意次序的选择：

① 如果存在程序的关键模块，在选择模块组装次序时，要使这些关键模块能尽早地组装进去。所谓关键模块，是指复杂的模块，或包含新算法的模块，又或者是怀疑有错误的模块。尽早组装关键模块可以尽早地发现关键错误，在进入组装前更改单元，另外早组装进入意味着在后续的组装测试中经受更频繁的考验。

② 在设计模块组装顺序时，要尽早使 I/O 模块加入序列，形成输入-处理-输出的骨架，这能使以后的测试工作简化，并减少测试的辅助性工作和人为因素所造成的测试错误和问题。

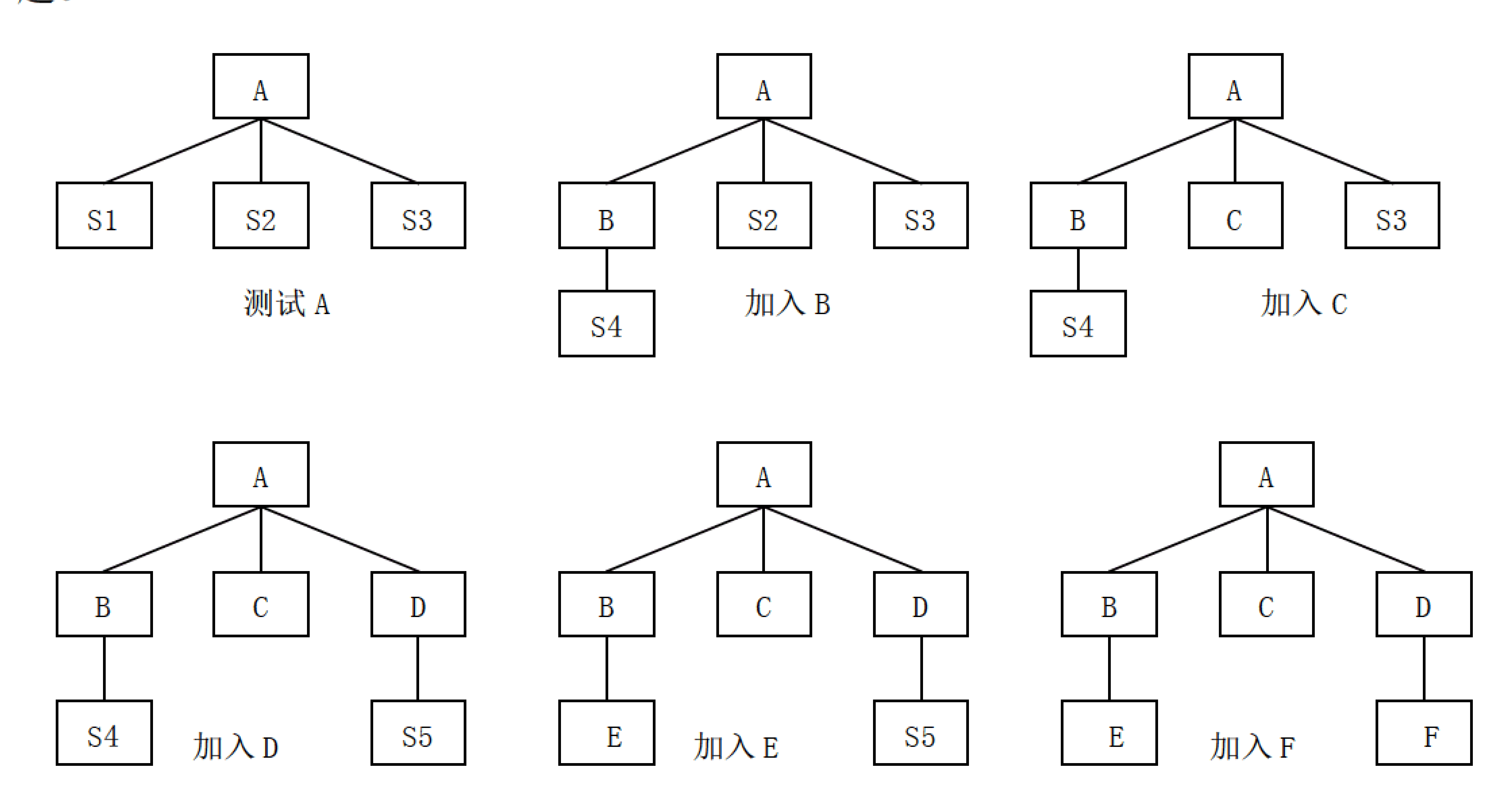


图 10-5　自顶向下按广度方向组装的例子

自顶向下组装测试的具体步骤为：①以主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代；②依据所选的集成策略(深度优先或广度优先)，以及新模块的选择原则，每次用一个实际单元替换被调用的一个桩模块，并开发该单元可能需要的桩模块；③每集成一个模块的同时立即进行测试，排除组装过程中可能引入的错误，如果测试发现错误，则要在修改后进行回归测试；④判断系统的组装测试是否完成，若没有完成，则转到②循环进行，直到集成结束。

自顶向下组装测试的优点是：①它在测试过程的早期，对主要的控制点或判决点进行检验。在分解得很好的软件结构中，判决需要在结构层次的较高层确定。如果主要的控制点有问题，早点认识到这个问题就变得很重要；②选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能，可先对逻辑输入的分支进行组装和测试，检查和克服潜藏的错误和缺陷，验证其功能的正确性，为此后主要分支的组装和测试提供保证；③能够较早地验证功能可行性，给开发者和用户带来成功的信心；④只有在个别情况下，才需要驱动程序(最多不超过一个)，减少了测试驱动程序开发和维护的费用；⑤可以和开发设计工作一起并行执行集成测试，能够灵活地适应目标环境；⑥容易进行故障隔离和错误定位。

自顶向下组装测试的缺点是：①在测试时需要为每个模块的下层模块提供桩模块，桩模块的开发和维护费用大；②底层组件的需求变更可能会影响到全局组件，需要修改整个

系统的多个上层模块；③要求控制模块具有比较高的可测试性；④在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，可能导致测试不充分。

解决这个问题有两种办法：①把某些测试推迟到用真实模块替代桩模块之后进行；②开发能模拟真实模块的桩模块。

自顶向下组装测试的适用范围是：①控制结构比较清晰和稳定的应用程序；②系统高层的模块接口变化的可能性比较小；③产品的低层模块接口还未定义或可能会经常因需求变更等原因被修改；④产品中的控制模块技术风险较大，需要尽可能提前验证；⑤需要尽早看到产品的系统功能行为；⑥在极限编程(Extreme Programming)中使用测试优先的开发方法。

2) 自底向上集成测试方法

自底向上集成测试是从程序模块结构的最底层的单元开始(即这种单元不再调用其他单元)，此后选择下一个单元时就不存在唯一的方法了。其中唯一的原则是，有资格作为下一个被选择的单元满足以下条件：这些单元的全部下层单元在此前已经被测试过。自底向上的测试仅需要对每个被测模块构造一个驱动模块，而不需要构造桩模块。

自底向上集成测试的具体步骤为：

(1) 开发一个测试驱动模块，由测试驱动模块控制最底层模块的并行测试；也可以把最底层模块组合成实现某个子功能的模块群，由测试驱动模块控制它进行测试。

(2) 用真实模块代替测试驱动模块，与它已经通过测试的下属模块组装成为完成更大功能的新模块群。

(3) 判断程序组装的过程是否已经达到主模块，如果是，则代表完成组装，结束测试，否则从(1)开始循环执行，直到组装结束。

以图 10-2 所示的系统结构为例，用图 10-6 来说明自底向上组装测试的顺序。

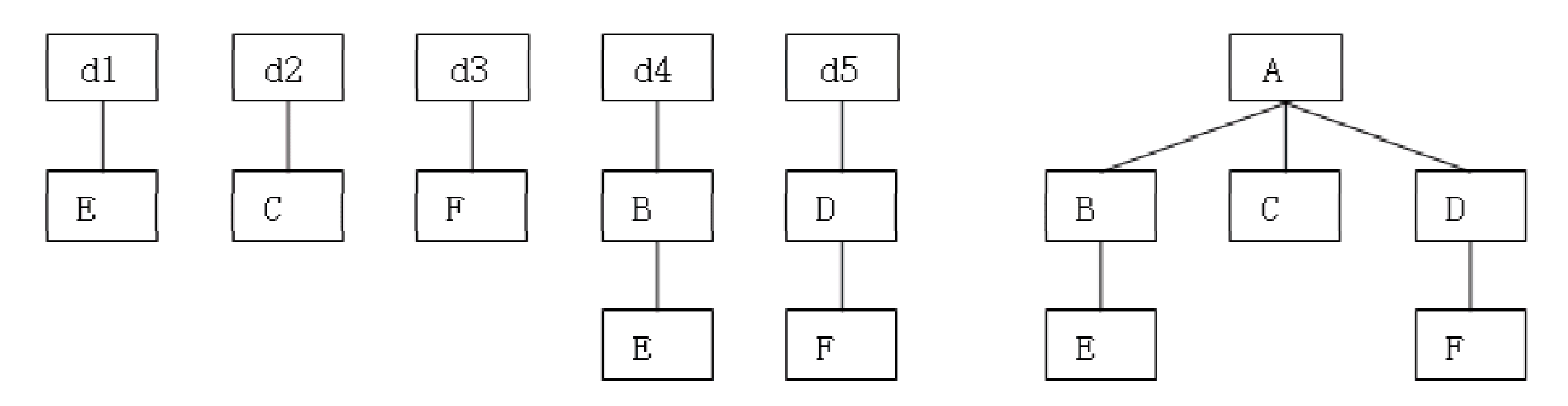


图 10-6　自底向上组装的例子

自底向上集成测试的优点在于：①由于驱动模块模拟了所有调用参数，测试模块返回的结果不影响驱动模块，生成测试数据也没有困难；②可以尽早地验证底层模块的行为，如果关键模块是在结构图的底部，自底向上集成测试是有优越性的；③自底向上的组装测试不必开发桩模块，提高了测试效率；④对实际被测模块的可测试性要求要少，容易对错误进行定位。

自底向上集成测试的缺点是：①当最后一个模块尚未测试时，还没有呈现出被测软件系统的雏形；②只有到测试过程的后期才能发现时序问题和资源竞争问题；③驱动模块的设计工作量大；④不能及时发现高层模块设计上的错误。

因此，在测试软件系统时，应根据软件的特点和工程的进度，选用适当的测试策略，有时混合使用两种策略更为有效。

自底向上集成测试的适用范围是：①底层模块接口比较稳定的产品；②高层模块接口变更比较频繁的产品；③底层模块开发和单元测试工作完成较早的产品。

3) 混合渐增式集成测试方法(或称三明治集成方法)

自顶向下集成测试方法和自底向上集成测试方法各有优缺点，一般来讲，一种方法的优点是另一种方法的缺点，因此产生了混合渐增式集成测试方法。下面介绍三种常见的混合渐增式集成测试方法。

(1) 衍变的自顶向下渐增式集成测试，它的基本思想是强化对输入/输出模块和引入新算法模块进行测试，再自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行渐增式集成测试。

(2) 自底向上-自顶向下的渐增式集成测试，首先对含读操作的子系统自底向上直至根节点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

(3) 回归测试，这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否匹配。

在组装测试时，测试者应当确定关键模块，对这些关键模块及早进行测试(对该模块及其所在层下面的各层使用自底向上的集成策略，然后对该模块所在层上面的各层使用自顶向下的集成策略，最后对系统进行整体测试)。关键模块至少应具有这几种特征：①满足某些软件需求；②在程序的模块结构中位于较高的层次(高层控制模块)；③较复杂，较容易发生错误；④有明确定义的性能要求。

混合渐增式集成测试方法的优点是：除了具有自顶向下和自底向上两种集成策略的优点之外，运用一定的技巧，能够减少桩模块和驱动模块的开发。缺点则是在被集成之前，中间层不能尽早得到充分的测试。

混合渐增式集成测试方法的适用范围：多数软件开发项目都可以应用此集成测试策略。

现在我们已经分别介绍了非渐增式集成测试方法和渐增式集成测试方法，从中可以看出渐增式集成测试方法相比非渐增式测试方法有以下优点：

(1) 非渐增式测试集成需要更多的工作量，对于程序模块结构，使用非渐增式集成测试方法可能需要五个驱动模块和六个桩模块。但对自底向上的渐增式集成测试方法仅需要五个驱动模块，不需要构造桩模块，减少了辅助性测试工作。

(2) 非渐增式集成测试方法先分散测试，再集中起来一次完成组合和测试。如果在模块接口处存在差错，只会在最后组合时一下子暴露出来。而使用渐增式集成测试方法可以较早地发现模块接口错误，这是由于较早地把模块组合起来进行测试所致。

(3) 作为结果，使用渐增式集成测试将使调试工作变得容易，因为渐增式集成测试逐步组合和逐步测试模块，把可能出现的错误逐步分散暴露出来，并且由于每次组合一个模块，错误发生时，比较容易地定位；这些错误是在最新增加的模块的连接中出现的。反之，使用非渐增式集成测试方法，直到对各个模块测试结束，对整个程序进行组合时才能发现错误，这时再确定错误的位置就非常困难，因为它可能出现在程序的任何地方。

(4) 渐增式集成测试方法利用以前已测试过的模块取代非渐增式集成测试方法中所需

要的驱动(或桩)模块，这样对后面模块的测试会使得前面已经实际测试过的模块得到更多的检验，因而使得整个程序的测试能取得较好的效果。

那么非渐增式测试方法为什么还要存在呢？一个原因是，在实际工作中有人是这样进行程序的组装测试的，需要在这里指出其弊端；另一个原因是，非渐增式集成测试方法在特定条件和特定范围内能起到一定作用，把整个软件系统组装起来也很快，但是必须小心谨慎。非渐增式集成测试方法的应用必须具备一系列的条件，但这仅仅是必要条件。那就是在一个做得很好且高度模块化的设计中，模块间的相互作用很小，而且详尽说明了接口，且将接口错误保持在最低限度，这时可以考虑用非渐增式集成测试方法。

10.1.3 基于 McCabe 的设计复杂性与集成复杂性的集成测试方法

集成复杂性的特性包括三点：①S1可以被用于限定一个项目的集成测试；②S1确定了集成测试基本集的数目；③每 S1个测试确认几个模块的集成。

在测试方法学的发展中，这些特性中的每一个都起着重要的作用。

1. 设计复杂性的特性

作为度量尺度，设计复杂度 S0 展示了一些重要特性。首先，S0 的上下限为：

$$n \le S_0 \le \sum_{i=1}^{n} v(G_i) \qquad \text{其中 n 为设计中的模块数}$$

当一个设计的 S0=n 时，其理由是相同的，即没有对下属模块的条件调用，因而每一模块的模块设计复杂度 iv(G)=1。另一方面，如果设计中的每一分支都影响模块的流向，那么 S0 等于所有模块的复杂度之和。

设计谓词这一概念可以用于计算设计复杂度。在设计阶段，既然结构图只是为了分析而构造的，那么用设计谓词来限制设计复杂度是一种可行的方法。使用这种方法，不需要考虑每一模块的内部伪代码，就可以评估出该设计。

通常约定，用点来表示从上层模块对下层模块的条件调用，用弧来表示从上层模块对下层模块的重复调用。这时，我们把条件和重复定义为设计谓词，而且模块 M 的模块设计复杂度 iv(M)等于设计谓词数加一。

设计复杂度的几个附加特性：

① 在设计中增加一个模块至少将 S0 加 1。

② 在调用模块时增加一个判定将使 S0 加 1。

③ 对模块 M(可重用代码)的多次调用使 S0 减少 S0(M)。

④ 对于任何子设计 D(模块 M 及其后代)，S0 被定义为：

$$S_0(D) = \sum iv(G_i) \qquad i \in D$$

当两个子系统 A 和 B 被集成为一个较大的系统 M 时：

模块 M 由 A 与 B 的并(∪)确定，表示为 M=A∪B。

模块数目(用 n(M)表示)为：n(M)=n(A)+n(B)−n(AΩB)；Ω表示交集，n(AΩB)是 A 和 B 共同拥有的模块数目。

设计复杂度 S0(M)为：S0(M)= S0(A) + S0(B) – S0(AΩB)；S0(AΩB)是 A 和 B 共同拥有模块的设计复杂度。

集成测试数 S1(M)= S1(A) + S1(B) – S1(AΩB)；S1(AΩB)是 A 和 B 共同拥有模块的集成复杂度。

2. 结构化集成测试方法学

结构化集成测试利用开发的设计尺度，产生一个可从设计规范说明中得出的测试策略。该方法在两个层次上适用：模块集成测试和设计集成测试。

模块集成测试的适用范围为一个模块及紧随其后的下属模块，当把多个模块打包为一个程序时，该测试尤为重要，通常该测试分三步实现：①将简化规则应用于被选中的模块；②简化后的子算法的圈复杂度是原始算法的模块设计复杂度，模块设计复杂度确定了该模块与其直接下属模块集成时所需的模块集成测试的数目；③对该子算法使用基线方法产生设计子树和模块集成测试。

第二层次的测试——设计集成测试来源于集成复杂度，因为集成复杂度确定了集成测试基本集合的数量，所以它还可以用来建立设计的集成测试策略。因而，它确定了测试一个设计的工作量。该方法的使用步骤如下：①计算每一模块的 iv；②计算每一模块的 S0；③用最顶层模块计算 S1(S1 是用于限定设计的基本子树的数目)；④建立一个 S1*n 的路径矩阵以用于基本子树集合；⑤在设计树上为每一谓词标识上标号，将谓词标号放在路径矩阵中该谓词影响到的模块所对应列的上方；⑥使用基线方法对矩阵进行操作，用 1 表明模块已被执行，用 0 表明模块未执行；⑦标识出矩阵中的每一子树；⑧标识出驱动子树的条件；⑨为每一子树建立相应的测试用例。

为了说明整个方法，考虑图 10-7 的设计结构，该设计由六个模块组成，并包含两个设计谓词。由于模块 D 是共享模块，因此设计结构不能累加，采用步骤①到步骤③之后，得到设计的 iv、S0 和 S1，设计复杂度为 8，集成复杂度为 3。

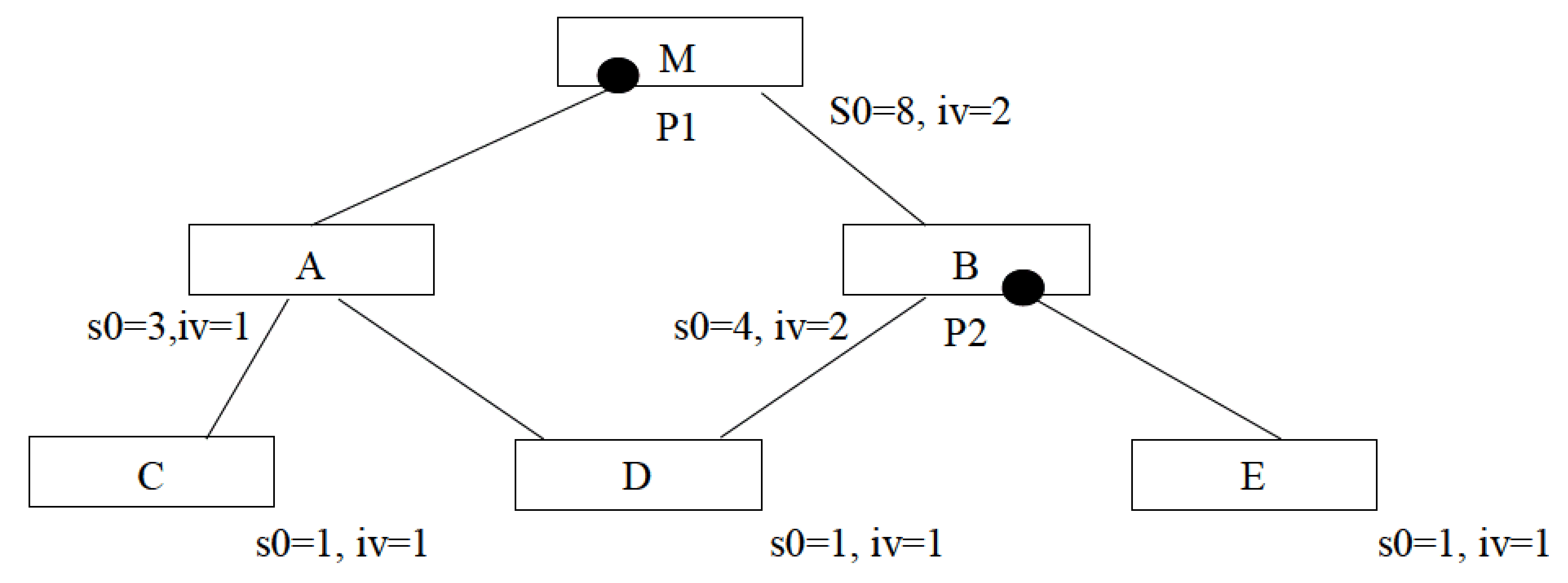


图 10-7　设计集成测试的例子

集成测试的路径矩阵(步骤④和步骤⑤)被创建为如图 10-8 所示的 3×6 矩阵，在此例中，两个设计谓词被标为 P1 和 P2，它们被放在模块 A 和 E 的上方，因为它们影响这两个模块的执行。采用基线方法，有关能执行系统中所有模块的基线被选择，随后 1 被放在每一模块下以表明此模块被执行，通过求基线上谓词的值(通常自左往右)来发现替换子树。

当求谓词 P1 的值时，条件是否定的，因为 P1 前面已经被设置为执行模块 A，所有现在设置为不执行模块 A、从属于模块 A 的每一模块都将不执行。第三个子树，即谓词 P2 被求值，在本例中，它也被设置为在基线时被执行，通过对 P2 求反及标识出它的从属模块，可以定义第三棵树。最终，路径矩阵如图 10-8 所示。

		P1				P2		
路 径	M	A	B	C	D	E	测试条件	所期望的执行
基线	1	1	1	1	1	1	W=X and Y=Z	调用 A 和 E
子树 2	1	0	1	0	1	1	W≠X and Y=Z	调用 E，不调用 A
子树 3	1	1	1	1	1	0	W=X and Y≠Z	调用 A，不调用 E
相对频度	3	2	3	2	3	2		

图 10-8　集成路径测试矩阵

子树被标识，而且驱动子树的条件也通过使用路径矩阵建立了，假定 P1 和 P2 分别是 W＝X 和 Y＝Z 这样的简单谓词，那么该设计的集成测试需求将包含三个具有相应条件的子树(步骤⑦和⑧)。

正是在体系结构级别，设计尺度可以提供额外的开发支持。从历史上看，有几种设计方法，比如自顶向下、自底向上、关键部分第一以及其他方法，这些方法可以被用来评估测试需求、建立测试计划以便支持所选择的设计方法。我们可以回到图 10-7 所示的例子上，假定使用关键部分第一设计方法选择基线，由于实时系统的性能需要，假定模块 E 是关键模块，基线被选择要尽可能地隔离模块 E，只包含那些无条件的模块及模块 E 的从属模块，采用着重考虑模块 E 这种测试方法，产生了另一个路径矩阵，如图 10-9 所示。

		P1				P2
路 径	M	A	B	C	D	E
基线	1	0	1	0	1	1
子树 2	1	1	1	1	1	1
子树 3	1	0	1	0	1	0
相对频度	3	1	3	1	3	2

图 10-9　关键模块集成测试矩阵

注意在图 10-9 中，测试过程中模块执行的相对频度是不同的，在这个简单的例子中，模块 A 和 C 不被经常执行。对于各种给定的设计方法，基线的建立都要把精力放在执行中关键的部分，从而软件设计者可以采用一种最适用于所选择的设计方法且能从设计说明中推导出来的设计策略。

从相对频度中，我们可以看出哪些模块经常被执行，哪些模块不经常被执行。至于基线的建立，可以在实际测试中针对关键成分来选择。

3. 小结

对设计复杂性的度量为系统开发者提供了一个尺度，这一尺度可以作为重要的管理工具。我们的设计尺度——模块设计复杂度、设计复杂度和集成复杂度，就是三个这样的度量值，系统设计者可以通过三个设计尺度来度量软件设计的复杂度。这些尺度是从圈复杂性派生出来的。由于程序中的判定结构是程序复杂性的重要象征，因此指明了设计中各模块之间关系的设计结构也就定义了总的设计复杂性。我们测试的是模块之间是如何在一起

工作的，而非单一模块是如何工作的，用于确定设计复杂性的简化技术强调的正是这一测试需求。更重要的是，这一度量尺度可以在两个层次上用于驱动测试过程：单独的模块和总设计框架。设计尺度的计算表明了一个新的管理和测试工具，这一工具对于开发者来说，以前是难以得到的。

- 直观上，设计复杂性度量展现了许多期望的特性，这些特性支持它们的适应性。
- 这一度量直观上与设计的理解难度相关，当我们检查一个复杂的大型设计时，该度量应该产生一个高值。直观上，我们认为简单的设计应该有相应低的值。
- 这一度量除了直观外，还应该是客观的，而且在数学上很严格。这样就能够在不同的时间或两个不同的人看同一设计产生相同的复杂度。如果这一度量不客观，那么在开发工作中涉及的各种既定的问题将毫无疑问地有不同的解释。
- 这一度量应该与集成该设计的工作量相关。与设计相关的最昂贵的活动是将各模块及子系统集成的工作，这一度量应该与集成阶段所花费的财力和精力直接相关。
- 这一度量应该是可用的。能够预测成本和错误这类信息的度量是有用的。另外，如果这一度量能够直接驱动设计模块化过程和集成过程，那么对于软件开发者来说，它是非常有意义的，而且对管理者和质量保证人员会产生很大的吸引力。
- 这一度量有助于在生命周期的前期产生一个集成测试计划。如果这一度量可以在设计阶段计算出来，那么来源于设计的测试集可以以严格的方式贯穿体系结构。设计错误比代码错误要昂贵 30 倍，所以早期捕捉到这些错误将节省大量的成本。
- 这一度量及其相应过程是自动的，随着 CASE 软件的迅猛发展，模块设计复杂度、设计复杂度、集成复杂度及测试开发过程可以自动进行，集成过程变得相当可行。实际上，McCabe & Associates 公司已经开发了一个工具，该工具可以计算设计程度并产生集成子树，这一自动工具目前已经被成功地用于几个项目中。

10.1.4　集成测试过程

集成测试是一种正规测试过程，有着不同阶段的任务。集成测试通常划分为 4 个阶段：集成测试计划阶段、集成测试设计与开发阶段、集成测试执行阶段、集成测试评估阶段。

集成测试通常是在开始进行体系结构设计的时候介入并制定测试方案，在进入详细设计之前完成集成测试方案，在进入系统测试之前结束集成测试。

1. 集成测试各阶段任务的划分

1) 集成测试计划阶段

一般安排在概要设计评审通过后大约一个星期的时候，参考需求规格说明书、概要设计文档、产品开发计划时间表来制定。

集成测试计划包含的内容有：①确定集成测试策略、方法、内容、范围、通过准则；②工具考虑，复用分析；③基于项目人力、设备、技术、市场要求等各方面决策；④集成测试进度计划；⑤工作量估算、资源需求、进度安排、风险分析和应对措施；⑥集成测试方案编制；⑦接口分析、测试项、测试特性分析(要体现测试策略)。

编制集成测试计划要与单元测试的完成时间协调起来，要考虑如下因素：①是采用何种系统组装方法来进行组装测试；②组装测试过程中要考虑集成的层次、软件的层次、模

块的复杂度和重要性以及连接各个模块的顺序；③模块代码编制和测试进度是否与组装测试的顺序一致；④测试过程中是否需要专门的硬件设备。

解决了上述问题之后，就可以列出各个模块的编制、测试计划表，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期，以及需要的测试用例和期望的测试结果。

在缺少软件测试所需要的硬件设备时，应检查该硬件的交付日期是否与集成测试计划一致。例如，若测试需要数字化仪和绘图仪，则相应测试应安排在这些设备能够投入使用之时，并需要为硬件的安装和交付使用保留一段时间，以留下时间余量。此外，在测试计划中需要考虑测试所需软件(驱动模块、桩模块、测试用例生成程序等)的准备情况。

2) 集成测试设计与开发阶段

一般在详细设计开始时，就可以着手进行。可以把需求规格说明书、概要设计、集成测试计划文档作为参考依据。

此时要完成测试规程/测试用例的设计与开发，确定测试步骤，设计测试数据，完成测试工具、测试驱动和桩的开发。

3) 集成测试执行阶段

当所有的集成测试工作准备完毕后，测试人员在单元测试完成以后就可以执行集成测试。

此时要搭建好测试环境，开展测试工作，确定测试结果，处理测试过程中的异常。

4) 集成测试评估阶段

当集成测试执行结束后，要召集相关人员对测试工作进行度量，对测试结果进行评估，确定是否通过集成测试。

执行阶段的度量要采集相关数据，包括：集成测试对象的数量、运行的用例数量、通过/失败的用例数量、发现的缺陷数量、遗留的缺陷数量、集成测试执行的工作量。

评估阶段要完成的工作有：按照集成测试报告模板出具集成测试报告，如有必要，对集成测试报告进行评审，将所有测试相关工作产品纳入配置管理。

2. 集成测试工作开展的原则

集成测试可以由开发部门进行，也可以由独立的测试部门执行。开发部门尽量进行集成测试，测试部门可有选择地进行集成测试。

在开展集成测试时，要遵循如下原则：

(1) 集成测试是产品研发中的重要工作，需要为其分配足够的资源和时间。

(2) 集成测试需要经过严密的计划，并严格按计划执行。

(3) 应采取增量式的分步集成方式，逐步进行软件部件的集成和测试。

(4) 应重视测试自动化技术的引入与应用，不断提高集成测试效率。

(5) 应该注意测试用例的积累和管理，方便进行回归并进行测试用例补充。

怎样有效、合理地开展集成测试，即选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号和测试的次序、生成测试用例和调试的费用等，这些问题在集成测试开展过程中都应该考虑。

3. 集成测试工作完成的标志

怎样判定集成测试过程完成了？可按以下几个方面检查：

(1) 成功地执行了测试计划中规定的所有集成测试。

(2) 修正了发现的所有错误。

(3) 测试结果通过专门小组的评审。

集成测试应由专门的测试小组来进行，测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。

在完成预定的组装测试工作之后，测试小组应负责对测试结果进行整理、分析，形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。此外还应提出目前不能解决，还需要管理人员和开发人员注意的一些问题，提供测试评审和最终决策，以提出处理意见。

10.2　确认测试

经集成测试后，已经按照设计把所有的模块组装成一个完整的软件系统，各单元之间的接口错误也已经基本排除了，接着就应该进一步验证软件的有效性，这时测试工作进入确认阶段(或称配置项测试)。

在一些软件测试阶段划分模型中，确认测试是介于集成测试和系统测试之间必不可少的一项测试工作或任务，也是软件测试分级中的一个重要级别。

10.2.1　确认测试的基本概念

确认测试是严格遵循有关标准的一种符合性测试，以确定软件产品是否满足规定要求。若能达到这一要求，则认为开发的软件是合格的。因而有时又将确认测试称为合格性测试。所谓规定要求，指的是软件规格说明书中确定的软件功能和技术指标，或是专门为测试所规定的确认准则。

确认测试是在完成集成测试之后，依据确认测试准则，针对软件需求规格说明进行的测试，以确认所开发的软件系统能否满足规定的功能和性能需求。软件确认测试通常采用黑盒测试方法。测试内容主要包括安装测试、功能测试、性能测试、可靠性测试、安全性测试、效率测试、易用性测试、可移植性测试、可维护性测试、文档测试等。由于确认测试针对的是用户的直接需求，因此应在尽可能真实的环境中进行。

确认测试必须有用户的积极参与，或以用户为主进行。用户应该参与设计测试方案，使用用户界面输入测试数据并且分析评价测试的输出结果。为了使用户能够积极主动地参与确认测试，特别是为了使用户可以有效地使用这个软件系统，通常在验收之前由软件开发单位对用户进行培训。

在确认测试中，alpha 测试在开发场所进行、有用户参与，而 beta 测试是在客户场所进行的。

10.2.2 确认测试的过程

确认测试的执行过程必须依照相关的软件测试评价标准，制定相应的测试规范，利用确认测试可能的测试方法，借助可能的测试工具，逐一测试上述标准中所有的测试项。测试的具体组织与实施必须在已建立的软件测试管理体系下展开，通过确认测试确认软件是否满足用户需求，相关结论应以规范的测试报告形式给出。

在确认测试阶段需要做的工作如图 10-10 所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过专家鉴定之后，才能成为可交付的软件。

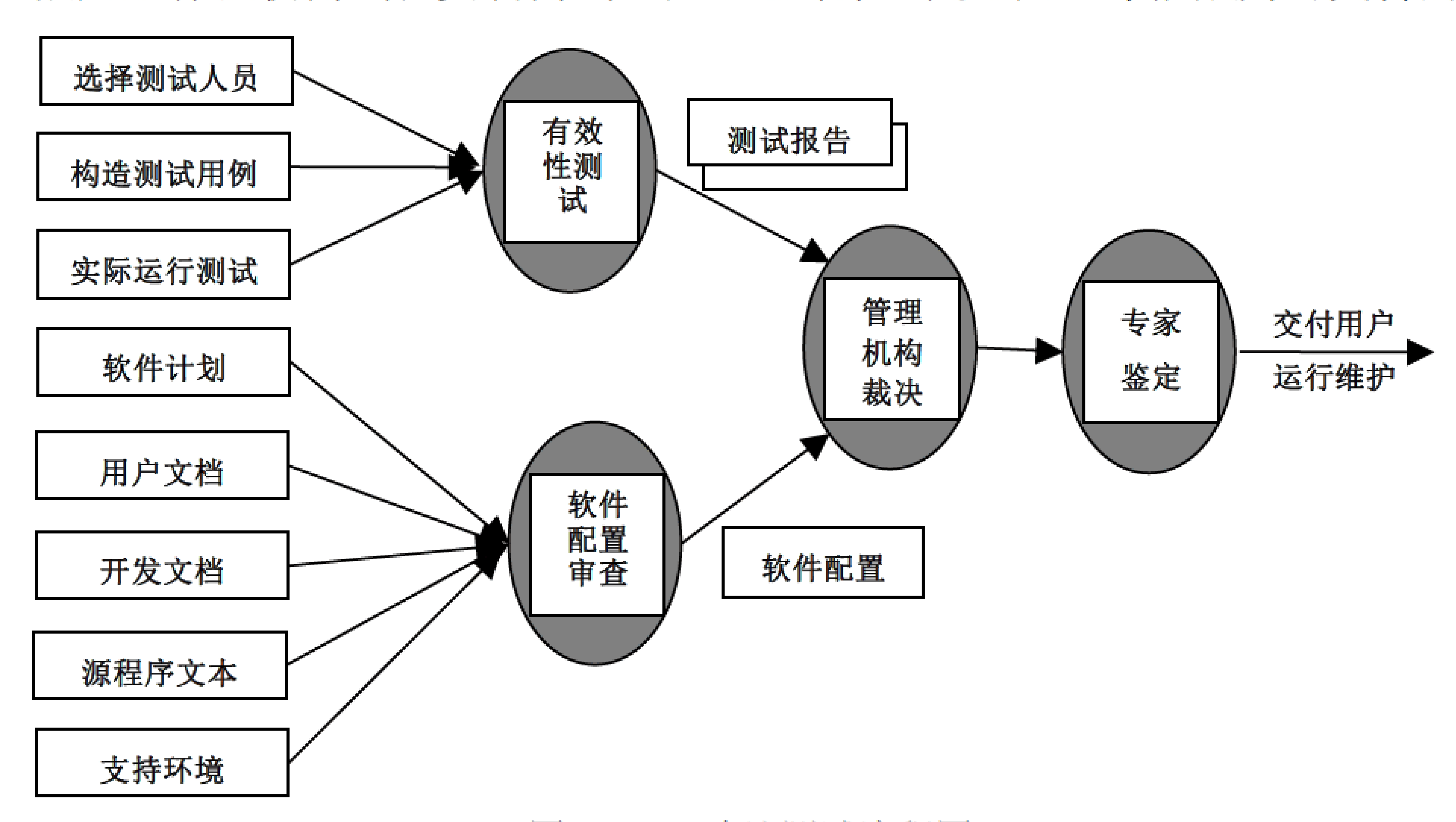


图 10-10 确认测试流程图

1. 有效性测试

确认测试又称有效性测试，这是因为在软件需求规格说明中，要求描述全部用户可见的软件属性，其中有一条叫作有效性准则，它包含的信息就是软件确认测试的基础。有效性测试是在模拟的环境下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书中列出的需求(主要任务是验证软件的功能和性能及其他特性是否与用户的要求一致)。软件的功能和性能要求在软件需求规格说明书中已经明确规定。为此，在确认测试过程中，应该仔细设计测试计划和测试过程，测试计划包括要进行的测试的种类及进度安排，测试过程规定了用来检测软件是否与需求一致的测试方案。通过测试和调试，要保证软件能满足所有功能要求，能达到每个性能要求，文档资料是准确而完整的，此外，还应该保证软件能满足其他预定的要求(例如，软件的安全性、可移植性、兼容性、可维护性等)。

在有效性测试中，当全部软件测试用例运行完毕后，会有下述两种可能的结果：

(1) 功能和性能与用户要求一致，这说明软件的这部分功能或性能特征与需求规格说明书相符合，从而接受这部分程序。

(2) 功能和性能与用户要求有差距，这说明软件的这部分功能或性能特征与需求规格说明书不相符合，因此要为它提交一份问题报告。

在这个阶段发现的问题往往和需求分析阶段的差错有关，涉及的面通常比较广，因此解决起来也比较困难。为了制定解决确认测试过程中发现的软件缺陷或错误的策略，通常需要和用户进行充分的协商。

2. 软件配置复查

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。其主要内容为文档资料，如用户所需资料(如用户手册、操作手册)、设计资料(如设计说明书等)、源程序以及测试资料(如测试说明书、测试报告等)。配置审查(configuration review)有时也称配置审计(configuration audit)。

除了按合同规定的内容和要求，由人工审查软件配置外，在确认测试过程中还应该严格遵循用户指南及其他操作程序，以便检验这些使用手册的完整性和正确性，且必须仔细记录发现的遗漏或错误，并适当地补充和改正。

3. α 测试和 β 测试

在软件交付使用之后，用户将如何使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误解、异常的数据组合，以及产生对于某些用户来说似乎是清晰的，但是对于另一些用户来说确却是难以理解的输出，等等。

如果软件是专为某个客户开发的，可以进行一系列验收测试，以便用户确认所有需求都得到满足。验收测试是由最终用户而不是系统的开发者进行的。事实上，验收测试可以持续几个星期甚至几个月，因此能够发现随着时间流逝可能会降低系统质量的累积错误。

但如果软件是为多个用户开发的，让每个用户逐个进行正式的验收测试是不切实际的。那么在这种情况下，绝大多数软件开发商都使用被称为 α 和 β 测试的过程，来发现那些看起来只有最终用户才能发现的错误。

α 测试是一种由用户在开发环境下进行的测试，也可以是由开发机构内部的用户在模拟实际操作环境下进行的测试。软件在自然设置状态下使用。开发者在用户旁边，对用户进行指导，并负责记录发现的错误和使用中遇到的问题。这是在受控环境下进行的测试。α 测试的目的是评价软件产品的功能、可使用化、可靠性、性能和支持。尤其注重产品的界面和特色。α 测试人员是除开发人员外首先见到产品的人，他们提出的功能和修改意见是非常有价值的。

β 测试由软件的最终用户在一个或多个客户场所进行。与 α 测试不同，开发者不在 β 测试的现场，因此 β 测试是软件在开发者不能控制的环境中的真实应用。用户记录在 β 测试过程中遇到的一切问题，并且定期把这些问题报告给开发者。开发者在接收到 β 测试的测试报告后，对软件产品进行必要的修改，并准备向全体客户发布最终的软件产品。β 测试主要衡量产品的功能、可使用化、可靠性、性能和支持，注重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。

4. 验收测试

在通过系统的有效性测试及软件配置审查之后，就应该开始系统的验收测试。验收测试是以用户为主的测试。软件开发人员和软件质量保证人员也应该参加。由用户参与设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试，在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、

兼容性、可维护性、错误的恢复功能等进行确认。

5. 确认测试的结果

确认测试的结果分别是：①功能和性能与需求文档及用户的要求一致，软件可以接受，即通过测试；②功能和性能与需求文档及用户的要求有一定的差距，此时需要详细列出软件中的各项缺陷或问题的清单或列表，提交对应的问题报告。必要时，要与用户协商来解决所发现的缺陷或错误。

确认测试应交付的文档有：确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

10.3　集成测试应用举例

JUnit 集成测试

在单元测试环节，在尽量屏蔽模块之间相互干扰的情况下，重点关注模块内部逻辑的正确性。集成测试则是在将模块整合在一起后进行的测试，目的在于发现一些模块间整合的问题。有些功能很难通过模拟对象进行模拟，相反它们往往只能在真实模块整合后，才能真正运行起来。

1. 集成测试的关注点

(1) 集成测试是在系统即将开发完毕，对系统是否正常运作进行的测试。

(2) 主要针对每一个 package 类库单位下类的功能进行测试。

(3) 集成测试将系统当成一个黑盒子，仅关注系统的输出/输入。向客户提供质量保证，不专门对单个程序员进行评估。

(4) 集成测试不仅需要用到 JUnit 自动测试框架和编写测试代码，还需要更多人的协调，侧重点会放在业务逻辑的处理，同时考虑性能特点，以保证没有隐藏 bug 的出现。

(5) 集成测试的颗粒度主要放在整个系统上。

2. 案例模块功能

本例以高校学生选课管理系统中的注册和登录两个模块作为集成测试实例，模块功能流程如图 10-11 所示。

高校学生选课管理系统注册和登录模块是用户登录系统时首页上的两项功能。

(1) 注册功能

未注册用户单击“新用户注册”按钮后，进入新用户注册界面进行用户注册，内容填写完毕后单击“注册”按钮，注册成功。若注册成功，注册信息将被直接写入数据库，下次登录时自动匹配数据库信息，信息一致登录成功，否则登录失败。若注册信息时单击“重置”按钮，信息栏将清空，光标重新回到录入信息栏的第一行，不写入数据库。具体如图 10-12 和图 10-13 所示。

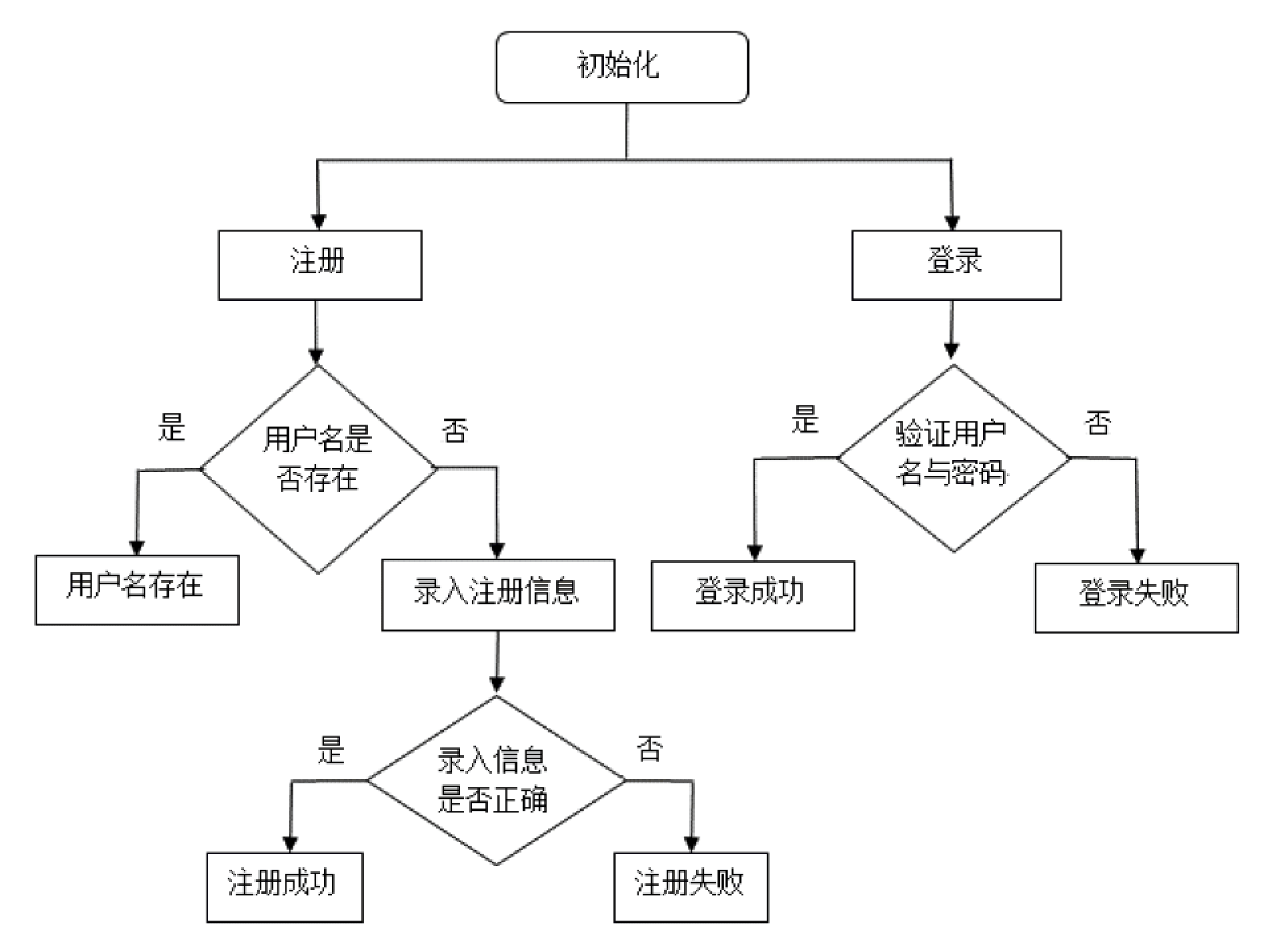


图 10-11　注册及登录模块功能流程图



图 10-12　高校学生选课管理系统的登录界面



图 10-13　高校学生选课管理系统的注册界面

(2) 登录功能

若登录用户已注册，正确输入用户名和密码，单击“登录”按钮，即可进入系统主页面。若输入错误，即输入信息与数据库匹配错误，登录失败。

3. 案例测试流程

使用 JUnit 进行集成，首先需要引用 JUnit 的 jar 包，可通过官网下载 jar 包，版本要与本地机器下载的 Spring 项目的版本号一致，否则会报错。

具体步骤及测试用例编写情况如下：

(1) @BeforeClass

该步骤可以对依赖的参数进行实例化操作，将参数信息加载到容器中，以便后续使用，此步骤是在测试类构造之前执行的方法。也就是说，使用此操作修饰的方法会在所有方法被调用前执行，而且该方法必须为 static void 类型，所以当测试类被加载后，就会运行@BeforeClass，针对所有测试，只执行一次，在内存中只会存在一份实例，比较适合加载配置文件、进行初始化等。实例代码如图 10-14 所示：

```java
@BeforeClass
public static void setUpBeforeClass() {
    System.out.println("=====初始化时实例化信息=======");
    context = new ClassPathXmlApplicationContext( configLocation: "/applicationContext.xml");
    userLoginDao = (IUserLoginDao) context.getBean( s: "userLoginDao");
}
```

图 10-14　初始化实例信息代码

(2) @Before

@Before 是初始化方法，该步骤是前置通知，也具有作为参数实例化的作用，对于每一个测试方法都要执行一次，并在所有方法执行前运行。实例代码如图 10-15 所示。

```java
/**
 * 前置通知信息
 */
@Before
public void setUp() {
    System.out.println("====前置通知：开始====");
}
```

图 10-15　前置通知信息代码

(3) @Test

@Test 是测试方法，该步骤作是测试类主要是获取信息的方法，运用接口调用已经写好的实现类，比如注册接口 RegLoginTest 调用注册实现类 regTest，确定此接口是否符合要求，并将输出信息打印，以便查看。实例代码如图 10-16 所示。

```java
@Test
public void regLoginTest(){
    regTest();
    if(regResult > 0){
        loginTest();
    }
}

/**
 * 注册
 */
private void regTest() {
    // 注册时判断用户名是否存在
    boolean isExist = userLoginDao.findByLoginName(loginName);
    if(!isExist){
        //实例化注册信息
        UserLogin userReg = new UserLogin();
        userReg.setLoginName(loginName);
        userReg.setPwd(pwd);
        userReg.setType("2");
        userReg.setMail("test@test.com");
        int regResult = userLoginDao.insert(userReg);
        if(regResult > 0){
            System.out.println("注册成功，进行登录");
        }else{
            System.out.println("注册失败，不进行登录");
        }
    }else{
        System.out.println("用户已存在，进行登录");
    }
}

/**
 * 登陆
 */
private void loginTest(){
    // 使用用户名和密码登陆
    userLoginDao.findbyNameAndPwd(loginName, pwd);
}
```

图 10-16　注册及登录模块测试用例信息代码

(4) @After

该步骤作为后置通知，也可以用于信息结果的后续判断处理。对于每一个测试方法都

要执行一次，在所有方法执行完毕后运行。实例代码如图 10-17 所示。

```
/**
 * 后置通知信息
 */
@After
public void tearDown() {
    System.out.println("====后置通知：结束====");
}
```

图 10-17　后置通知信息代码

(5) @AfterClass

该步骤作为最后结束的方法处理，针对所有测试，只执行一次，且必须为 static void 类型，在此测试类构造之后运行。即使用此操作所修饰的方法会在所有方法被调用后执行，通常用于资源的清理，如关闭数据库连接和不再使用的信息等。实例代码如图 10-18 所示。

```
@AfterClass
public static void tearDownAfterClass() {
    System.out.println("=====结束时将已经实例化的内容置为空=======");
    userLoginDao = null;
    context = null;
}
```

图 10-18　结束测试方法信息代码

(6) 测试结果

运行测试用例，查看测试结果，选中测试用例文件，选择 Run as | JUnit Test 运行程序，运行效果如图 10-19 所示：绿色表示测试用例运行通过，红色表示测试用例运行失败。

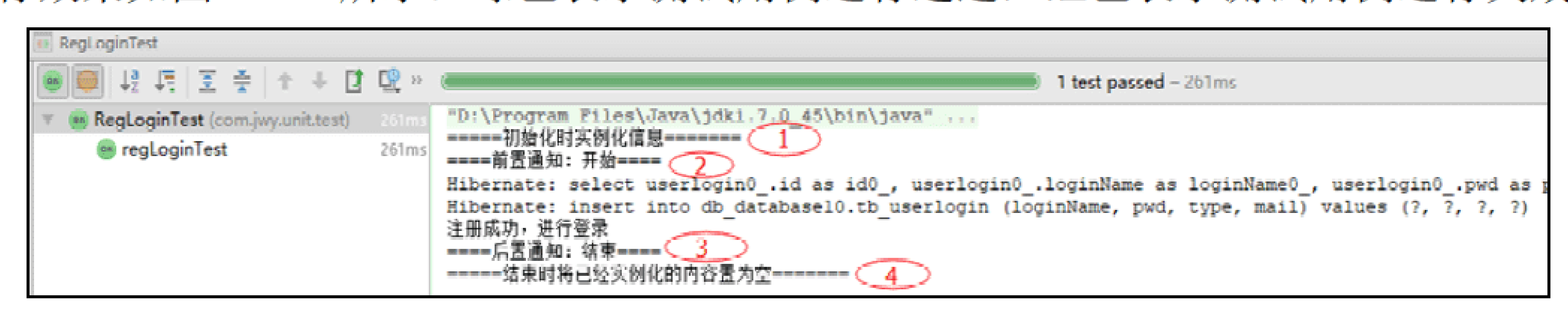


图 10-19　测试结果

习题和思考题

1. 什么是集成测试？集成测试采用的方法有哪些？进行集成测试时我们要考虑哪些问题？

2. 软件集成测试的作用是什么？集成测试与单元测试的异同点有哪些？

3. 在 Eclipse 环境下建立 Web 应用集成测试环境，并对具体的 Web 应用进行测试，详细描述测试过程。

4. 简述确认测试的概念。确认测试包括哪些内容？

第11章　软件系统测试

由于软件只是系统的一个组成部分，软件开发完成以后，最终还要与系统的其他部分配套运行，进行系统测试。系统测试是软件测试分级中一个非常重要的级别。下面的示例很好地说明了系统测试的重要性。

某无人机的飞控系统在系统测试中发现软件问题 50 个，其中关键错误 25 个：①当发动机空中停车后，系统不能进行发动机启动；②进入失速状态后，飞机失去控制；③链路中断又恢复，飞机不接受控制，不能着陆，越飞越远。

11.1　系统测试

11.1.1　系统测试的概念

什么是系统测试？不同的标准，定义也不一样，例如《软件工程术语》GB/T 11457-1995 给出的定义是——测试整个硬件和软件系统的过程，以验证系统是否满足规定的需求；《IEEE 软件验证与确认标准》1998 的定义是——为了验证和确认系统是否符合初始目标而对集成的软硬件系统进行的测试活动。

目前，软件工程界对系统测试的一般性定义为：系统测试是为判断系统是否符合规定而对集成的软硬件系统进行的测试活动。它将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行(使用)环境下，对计算机系统进行一系列的组装测试和确认测试。它是一个使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的系统需求或是弄清预期结果与实际结果之间的差别。

系统测试的对象是完整的、集成的计算机系统，重点是新开发的软件配置项的集合。因此，系统测试实际上是针对系统中各个组成部分进行的综合性检验。尽管每一个检验有着特定的目标，然而所有的检测工作都要验证系统中每个部分均已得到正确的实现，并能完成指定的功能。

系统测试的目的是在真实系统工作环境下通过与系统的需求定义(如功能需求)作比较，检验完整的软件配置项能否和系统正确连接，发现软件与系统/子系统设计文档和软件开发合同规定不符合或与之矛盾的地方，发现产品缺陷并度量产品质量。

另外，系统测试不仅关注系统的功能，也包括性能、安全等非功能测试。在实际的项目里，因为时间和投资预算的关系，测试资源主要消耗在功能测试上，非功能测试经常被很多项目忽略。这确实有点令人惋惜。尽管非功能测试的确非常难做而且大多数工作需要在项目开始的时候就开始做，但是认真对待非功能测试是非常有必要的。

最后，系统测试还要检验系统的文档等各种资料是否完整、有效。

对于软件工作而言，系统测试是软件研制人员参加系统的综合测试，将软件及计算机系统加入到系统中进行测试。应该一方面为系统测试提供必要的软硬件及资料支持，另一方面从软件测试角度提出系统测试中关于软件的测试设计。

系统测试的测试用例应根据需求分析说明书来设计，并在实际使用环境中运行。通常系统测试采用黑盒测试技术，并由独立的测试人员完成。

从软件测试角度看，系统测试有如下两方面的意义：

1) 系统测试的环境是软件真实运行环境的最逼真模拟

系统测试中，各部分研制完成的真实设备逐渐替代了模拟器，是软件从未有过的运行环境。有关真实性的一类错误，包括外围设备接口、输入/输出、多处理器设备之间的接口不相容、整个系统的时序匹配等，在这种运行环境中能得到比较全面的暴露。

2) 通常系统测试的困难在于不容易从系统目标直接生成测试用例

系统测试由系统人员组织，从系统完成任务的角度测试，软件在系统测试下获得了系统任务下直接的测试实例，这对检验软件是否满足系统任务要求是非常有意义的。

11.1.2　系统测试中关注的重要问题

按照软件测试生命周期概念，很显然，系统测试中最为关注的问题无非是系统测试过程定义、系统测试需求获取、系统测试策略选择、系统测试技术与方法、系统测试环境建立、系统测试人员组织以及系统测试要交付的文档等。下面我们就这些关注的问题进行分别论述。

1. 系统测试过程定义

系统测试过程与第 5 章叙述的测试过程是一致的，包括制定系统测试计划、设计测试系统、实施系统测试、执行系统测试和评估系统测试五个阶段，如图 11-1 所示。每个阶段的内容可参见表 11-1。

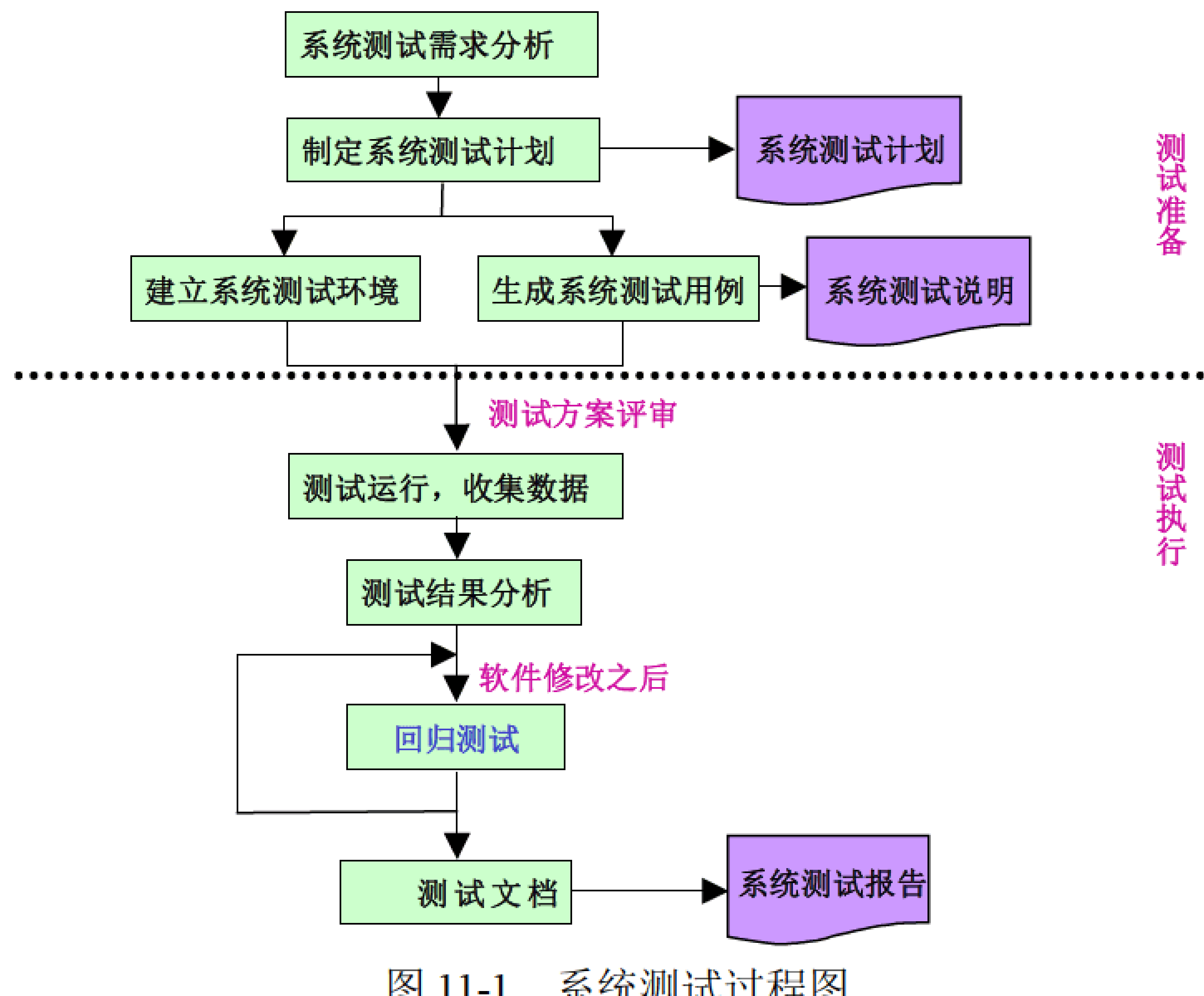


图 11-1　系统测试过程图

从图 11-1 可以看出，整个测试过程分为两个阶段：测试准备和测试执行。测试准备过程包括制定系统测试计划、建立系统测试环境、生成系统测试用例；测试执行过程包括测试运行、分析测试结果数据并生成软件问题清单、回归测试并生成测试报告。

表 11-1　系统测试各阶段的任务

活动名称	输入工作	输出工作	角色
制定系统测试计划	软件需求文档、软件项目计划	系统测试计划	测试设计员
设计系统测试	系统测试计划、软件设计文档	系统测试用例	测试设计员
实施系统测试	系统测试计划、系统测试用例	系统测试脚本	测试设计员
执行系统测试	系统测试计划、被测软件系统、系统测试用例、系统测试脚本	测试结果	测试员
评估系统测试	测试结果	软件测试报告	测试设计员、测试员

2. 系统测试需求获取

从软件的需求规格说明和其他相关文档中提取测试需求的过程，是一个寻找原子系统功能的过程(一种在系统层次上可以观察得到的端口输入和输出事件的行动)，系统测试需求主要来源于需求规格说明书或系统测试项目合同等。测试需求最终体现为测试定义、测试类型、测试内容及测试对象等。在进行系统测试需求分析时，可应用以下几条规则：

(1) 测试需求必须是可观测、可测评的行为。对于不能观测或测评的测试需求，是无法对其评估的，因此也就无法确定需求是否已经满足。

(2) 在每个用例或系统的补充需求与测试需求之间不存在一对一的关系。用例通常具有多个测试需求；有些补充需求将派生一个或多个测试需求，而其他补充需求(如市场需求或包装需求)将不派生任何测试需求。

(3) 在需求规格说明书中，每一个功能描述将派生一个或多个测试需求，性能描述、安全性描述等也将派生一个或多个测试需求。

(4) 在系统测试需求中，以传统测试类型中的功能性测试需求和性能测试需求最为重要，是整个系统测试需求的核心。

1) 功能性测试需求

功能性测试需求来自测试对象的功能性说明。每个用例至少会派生一个测试需求。对于每个用例事件流，测试需求的详细列表至少会包括一个测试需求。对于需求规格说明书中的功能描述，将至少派生一个测试需求。

2) 性能测试需求

性能测试需求来自测试对象的指定性能行为。性能通常被描述为响应时间和资源使用率的某种评测。性能需要在各种条件下进行评测，这些条件包括：①不同的工作量和/或系统条件；②不同的用例和功能不同的配置。

性能需求在补充规格或需求规格说明书中的性能描述部分说明。对包括这些内容的语句要特别注意：①时间语句，如响应时间或定时情况；②指出在规定时间内必须出现的事件数或用例数的语句；③将某一项性能的行为与另一项性能的行为进行比较的语句；④将

某一配置下的应用程序行为与另一配置下的应用程序行为进行比较的语句；⑤一段时间内的操作可靠性(平均故障时间 MTBF 或平均无故障时间 MTTF)；⑥配置或约束。

3) 其他测试需求

其他测试需求包括配置测试、安全性测试、容量测试、强度测试、故障恢复测试、负载测试等测试需求，可以从非功能性需求中发现与之对应的描述。每一个描述信息可以生成至少一个测试需求。

4) 系统测试策略选择

测试策略用于说明某项特定测试工作的一般方法和目标。系统测试策略主要针对系统测试需求确定测试类型以及实施测试的方法和技术。一个好的测试策略应该包括：要实施的测试类型和测试目标，采用的技术，用于评估测试结果和测试是否完成的标准，以及对测试策略所述的测试工作存在影响的特殊事项。

确定系统测试策略时，首先应清楚地说明所实施系统测试的类型和测试目标。清楚地说明这些信息有助于尽量避免混淆和误解(尤其是由于有些测试类型看起来非常类似，如强度测试和容量测试)。测试目标应该表明执行测试的原因。

系统测试的测试类型一般包括：功能测试、性能测试、负载测试、强度测试、容量测试、安全性测试、配置测试、故障恢复测试、安装测试、文档测试、用户界面测试等。其中，功能测试、配置测试、安装测试等在一般情况下是必需的，而其他的测试类型则需要根据软件项目的具体要求进行裁剪。

3. 系统测试技术与方法

系统测试主要采用黑盒测试技术设计测试用例，以确认软件满足需求规格说明的要求。

4. 系统测试环境建立

被测软件的可能的运行环境分别是开发环境、测试环境、用户环境。开发环境往往与用户环境有所差别；规划良好的测试环境总很接近于用户环境，但也要兼顾开发环境；测试环境在测试计划和测试用例中要事先定义和规划。

建立系统测试环境要考虑下列因素：

1) 确定硬件环境和软件环境

这里，硬件环境指测试必需的服务器、客户端、网络连接设备，以及打印机/扫描仪等辅助硬件设备所构成的环境，软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

2) 规划系统测试环境

分析用户环境中哪些配置可能对软件有所影响，并在此基础上规划和建立测试环境。

3) 在建立测试环境时需要考虑的因素

建立测试环境需要考虑计算机平台、操作系统、浏览器、软件支持平台、外围设备、网络环境、数据环境、其他专用环境等。

4) 确定建立系统测试环境的步骤

如安装应用程序、安装和开发测试工具、设置专用文件，包括将这些文件与测试所需的数据相对应、建立与应用程序通信的实用程序、配备适当的硬件以及必要的设备等。

5. 系统测试人员组织

系统测试至少需要由一个独立的测试组来开展工作，或者由项目组为每一个软件项目成立测试组，确定测试经理(通常由测试设计员担任)一名，测试设计员和测试员多名。

测试团队一般四人或五人，否则应该细分为测试组。测试经理/测试组长制定测试计划和测试方案，分配测试任务并检查测试进度，代表测试团队与开发、产品、用户沟通，开展实际测试，最后评估系统测试；测试设计员参与系统测试计划和方案的制定，按照系统测试方案设计测试用例和测试代码、设计所需测试工具、编写测试规程、完成系统测试报告并组织评审、输出测试案例和总结等经验文档；测试员执行系统测试用例、填写缺陷报告并检查缺陷处理结果。图 11-2 是系统测试过程中各阶段测试人员要做的工作。

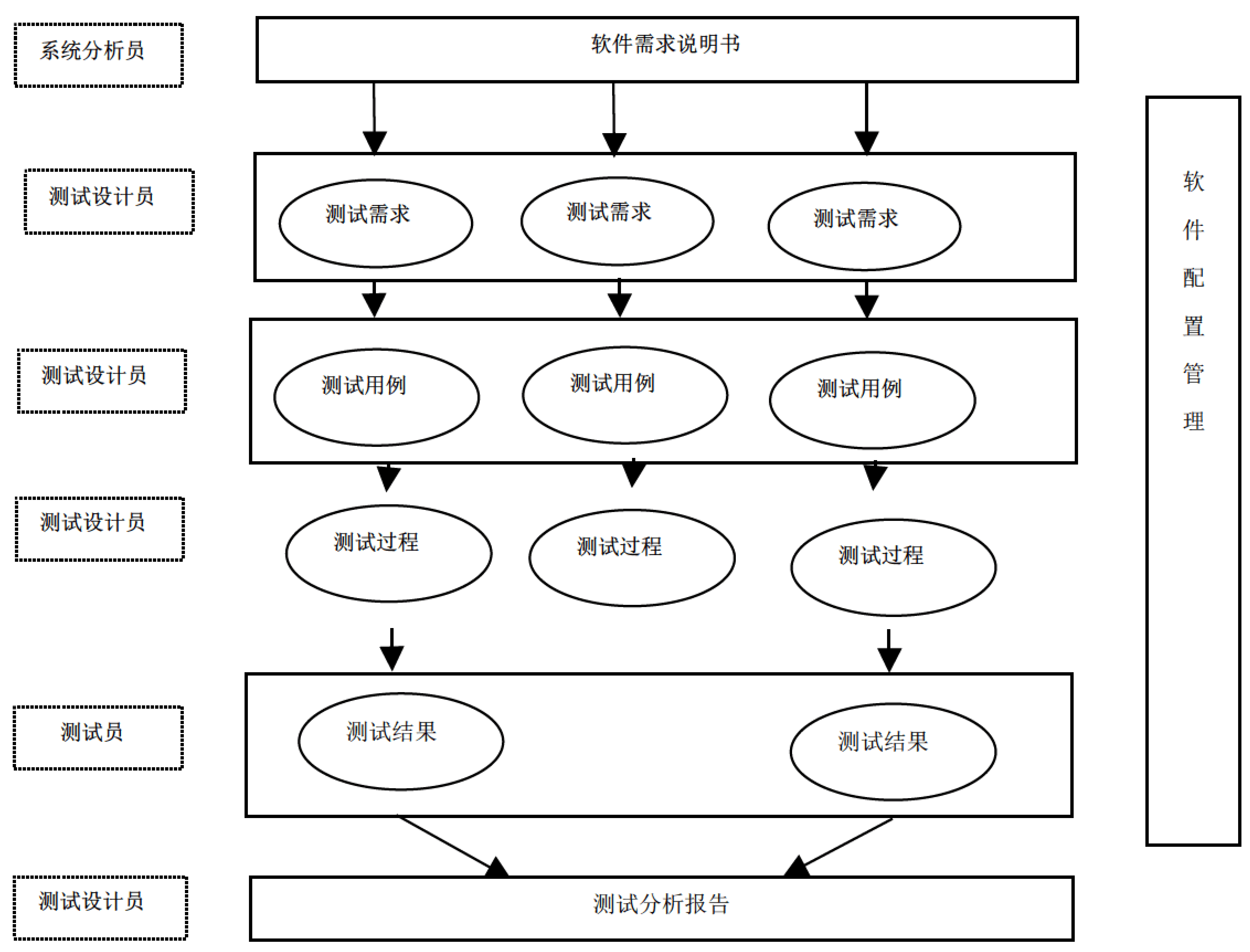


图 11-2　系统测试过程中各阶段测试人员要做的工作

根据相关统计数据我们知道，测试人员的效率是平均每个工作日发现 3 到 5 个错误；开发人员平均每修正 3 个错误，会引入 1 个新的错误；平均 75%的错误会在单元测试阶段解决；平均 20%的错误会在集成测试和系统测试阶段解决；平均 5%的 bug 会被交付给用户。普通大型民用软件平均错误率为 5 个/10000LOC，电信/银行/操作系统等软件的平均错误率 5 个/100000LOC。

软件测试与软件开发人员的配备与产品大小、复杂度、质量要求相关。目前国内外软

件测试与软件开发人员的比例相差很远。在软件产业发达国家，软件企业一般是把 40%的工作花在测试上，测试人员和开发人员之比平均在 1:1 以上，软件测试费用占整体开发费用的 30%~50%。对于要求高可靠性、高安全性的软件，测试费用则相当于整个软件项目开发所有费用的 3 至 5 倍。这些说明软件测试的重要性。而目前我国无论是政府、企业还是高等院校，对软件测试工作和人才培养一直不够重视，大家重开发、轻测试，致使我国在软件测试上的投入远远低于在软件开发上的投入，远远低于软件产业发达国家在软件测试上的投入。测试人员和开发人员之比平均在 1:6 或 1:8，甚至更低。

6. 系统测试要交付的文档

系统测试要交付的文档主要有：系统测试计划、系统测试计划评审报告、系统测试用例、系统测试用例评审报告、系统测试脚本、系统测试脚本评审报告、系统测试报告、系统测试报告评审报告、缺陷问题单若干等。

上诉报告可以根据需要进行裁剪或改造，如交付系统测试计划、系统测试设计、系统测试结果、缺陷问题等报告。

11.1.3　系统测试的要求和主要内容

现代系统测试要求依据软件质量特性/子特性来进行，重点是新开发的软件配置项的集合。但在实际测试中是针对传统测试中的各种测试类型(软件质量特性/子特性与软件测试类型的对应关系可参见表 6-1)，如功能测试、可靠性测试、性能测试、安全性测试、边界测试、余量测试、恢复性测试、接口测试和强度测试等。

1. 系统测试一般要求

系统测试一般应符合以下技术要求：

(1) 系统的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖。

(2) 测试用例的输入应至少包括有效等价类值、无效等价类值和边界数据值。

(3) 应逐项测试系统/子系统设计说明规定的系统的功能、性能等特性。

(4) 应测试软件配置项之间及软件配置项与硬件之间的接口。

(5) 应测试系统的输出及其格式；

(6) 应测试运行条件在边界状态和异常状态下，或在认为设定的状态下，系统的功能和性能。

(7) 应测试系统访问和数据安全性。

(8) 应测试系统的全部存储量、输入/输出通道和处理时间的余量。

(9) 应按系统或子系统设计文档的要求，对系统的功能、性能进行强度测试。

(10) 应测试设计中用于提高系统安全性、可靠性的结构、算法、容错、冗余、中断处理等方案。

(11) 对完整性级别高的系统，应对其进行安全性、可靠性分析，明确每一个危险状态和导致危险的可能原因，并对此进行针对性的测试。

(12) 对有恢复或重置功能需求的系统，应测试其恢复或重置功能和平均恢复时间，并且对每一类导致恢复或重置的情况进行测试。

(13) 对不同的实际问题应外加相应的专门测试。

对于具体的系统，可根据软件测试合同(或项目计划)及系统的重要性、完整性级别等要求对上述内容进行裁剪或修订。

2. 不同测试类型的测试要求

在系统测试中，对于具体的测试类型，测试内容及测试要求如下：

1) 功能测试

对软件需求规格说明中的功能需求逐项进行的测试，以验证其功能是否满足要求。

一般测试要求包括：用正常值的等价类输入数据值测试；用非正常值的等价类输入数据值测试；进行每个功能的合法边界值和非法边界值输入的测试；用一系列真实的数据类型和数据值运行，测试超负荷、饱和及其他最坏情况的结果；在配置项测试时对配置项控制流程的正确性、合理性等进行验证。

2) 性能测试

对软件需求规格说明中的性能需求逐项进行的测试，以验证其性能是否满足要求。

一般测试要求包括：测试在获得定量结果时程序计算的精确性(处理精度)；测试其时间特性和实际完成功能的时间(响应时间)；测试为完成功能所处理的数据量；测试程序运行所占用的空间；测试其负荷潜力；测试配置项各部分的协调性；在系统测试时测试软件性能和硬件性能的集成；在系统测试时测试系统对并发事务和并发用户访问的处理能力。

3) 接口测试

对软件需求规格说明中的接口需求逐项进行的测试。

一般测试要求包括：测试所有外部接口，检查接口信息的格式及内容；对每一个外部输入/输出接口必须执行正常和异常情况的测试；测试硬件提供的接口是否便于使用；测试系统特性(如数据特性、错误特性、速度特性)对软件功能、性能特性的影响；对所有内部接口的功能、性能进行测试。

4) 人机交互界面测试

对所有人机交互界面提供的操作和显示界面进行的测试，以检验是否满足用户的要求。

一般测试要求包括：测试操作和显示界面及界面风格与软件需求规格说明中要求的一致性和符合性；以非常规操作、误操作、快速操作来检验人机界面的健壮性；测试对错误命令或非法数据输入的检测能力与提示情况；测试对错误操作流程的检测与提示；对照用户手册或操作手册逐条进行操作和观察。

5) 强度测试

强制软件在不正常到发生故障的情况(设计的极限状态到超出极限)下运行，检验软件可以运行到何种程度的测试。

一般测试要求包括：提供最大处理的信息量；提供数据能力的饱和实验指标；提供最大存储范围(如常驻内存、缓冲、表格区、临时信息区)；在能力降级时进行测试；进行其他健壮性测试(测试人为错误下的反应，如寄存器数据跳变、错误的接口状态)；通过启动软件过载安全装置(如临界点警报、过载溢出功能、停止输入、取消低速设备等)生成必要

条件，进行计算过载的饱和测试。

6) 余量测试

对软件是否达到需求规格说明中要求的余量的测试。若无明确要求，一般至少留有20%的余量。

一般测试要求包括：全部存储量的余量；输入/输出及通道的余量；功能处理时间的余量。

7) 可靠性测试

在真实的或仿真的环境中，为做出软件可靠性估计而对软件进行的功能测试(其输入覆盖和环境覆盖一般大于普通的功能测试)。

在可靠性测试中必须按照运行剖面和使用的概率分布随机地选择测试用例。

8) 安全性测试

检验软件中已存在的安全性、安全保密性措施是否有效的测试。测试应尽可能在符合实际使用的条件下进行。

一般测试要求包括：对安全性关键的软件部件，必须单独测试安全性需求；在测试中全面检验防止危险状态措施的有效性和每个危险状态下的反应；对设计中用于提高安全性的结构、算法、容错、冗余及中断处理等方案，必须进行针对性测试；对软件处于标准配置下其处理和保护能力的测试；应进行对异常条件下系统/软件的处理和保护能力的测试(以表明不会因为可能的单个或多个输入错误而导致不安全状态)；对输入故障模式的测试；必须包含边界、界外及边界结合部的测试；对 0、穿越 0 以及从两个方向趋近于 0 的输入值的测试；必须包括在最坏情况配置下的最小输入和最大输入数据率的测试；对安全性关键的操作错误的测试；对具有防止非法进入软件并保护软件的数据完整性能力的测试；对双工切换、多机替换的正确性和连续性的测试；对重要数据的抗非法访问能力的测试。

9) 恢复性测试

对有恢复或重置(reset)功能的软件的每一类导致恢复或重置的情况，逐一进行的测试，以验证其恢复或重置功能。恢复性测试是要证实在克服硬件故障后，系统能否正常地继续进行工作，且不对系统造成任何损害。

一般测试要求包括：探测错误功能的测试；能否切换或自动启动备用硬件的测试；在故障发生时能否保护正在运行的作业和系统状态的测试；在系统恢复后，能否从最后记录下来的无错误状态开始继续执行作业的测试。

10) 边界测试

对软件处在边界或端点情况下运行状态的测试。

一般测试要求包括：软件的输入域或输出域的边界或端点的测试；状态转换的边界或端点的测试；功能界限的边界或端点的测试；性能界限的边界或端点的测试；容量界限的边界或端点的测试。

11) 数据处理测试

对完成专门数据处理功能所进行的测试。

一般测试要求包括：数据采集功能的测试；数据融合功能的测试；数据转换功能的测

试；剔除坏数据功能的测试；数据解释功能的测试。

12) 安装性测试

对安装过程是否符合安装规程的测试，以发现安装过程中的错误。

一般测试要求包括：不同配置下的安装和卸载测试；对安装规程的正确性的测试。

13) 容量测试

检验软件的能力最高能达到什么程度的测试。

要求测试到正常情况下软件的最高能力，如响应时间以及并发处理个数等。

14) 互操作性测试

为验证不同软件之间的互操作能力而进行的测试。

一般测试要求包括：必须同时运行两个或多个不同的软件；软件之间发生互操作。

15) 敏感性测试

为发现在有效输入类中可能引起某种不稳定性或不正常处理的某些数据的组合而进行的测试。

一般测试要求包括：发现有效输入类中可能引起某种不稳定性的数据组合的测试；发现有效输入类中可能引起某种不正常处理的数据组合的测试。

16) 标准符合性测试

验证软件与相关国家标准或规范(如军用标准、国家标准、行业标准以及国际标准)一致性的测试。

一般测试要求包括：建立标准符合性评价准则；逐一验证符合指定标准的能力。

17) 兼容性测试

验证软件在规定条件下与若干个实体共同使用或实现数据格式转换时能满足有关要求能力的测试。

一般测试要求包括：验证软件在规定条件下与若干个实体共同使用时满足有关要求能力；验证软件在规定条件下与若干个实体实现数据格式转换时能满足有关要求能力的测试。

18) 中文本地化测试

验证软件在不降低软件原有能力的条件下，处理中文能力的测试。

一般测试要求包括：测试软件使用中文的能力；测试软件处理中文的能力；测试软件兼容中文的能力；在中文环境下，对软件原有功能和能力的测试。

上述 17 种测试内容并不是都要进行，制定测试策略和测试计划的时候要有不同的侧重点，而这与测试目标、测试资源、软件系统特点和业务环境有关。

另外，上述 17 种测试最好由独立第三方进行测试。因为进行独立测试的目的是进一步加强软件质量保证工作，提高软件的质量，并对软件产品进行客观评价，而进行第三方独立测试通常有发挥专业技术优势和独立性优势，能够有效地促进承办方的工作等方面的优势。

11.1.4　系统测试设计

在系统测试需求确定后，可以开始测试设计工作。而测试设计又是整个测试过程中非常重要的一个环节，测试设计的输出结果是测试执行活动依赖的执行标准，测试设计的充分性决定了整个系统过程的测试质量。因此，为了保证系统测试质量，必须在测试设计阶段就对系统进行严密的测试设计。

测试设计的一般流程是：首先理解软件和测试目标，然后设计测试用例，接着运行测试用例并处理测试结果，最后评估测试用例和测试设计。

在确定测试设计流程时既强调目的性，也强调计划性，目标是追求测试的高效率和测试的理想结果。当然，水能载舟，亦能覆舟。文档化和按部就班方式可以降低管理难度，增强计划性，但也可能扼杀测试人员的经验作用和灵感。

1. 理解软件和测试目标

目的：建立软件故障模型，了解测试目标，确定测试策略和测试计划。

任务：①了解软件的功能和业务背景、用户环境；②了解软件的开发背景和系统结构、技术选型；③了解软件的质量历史、版本变化；④了解系统的测试目标和资源限制，确定测试策略和测试计划。

方法：①阅读软件使用手册，理解软件运行环境和用户行为，了解同类软件的功能和使用，了解软件要解决的问题域和解域(业务背景知识)；②试运行软件，从中熟悉软件功能，确定软件基本可以测试；③了解软件体系结构、技术选型、开发环境和工具；④阅读早期版本测试报告，以及单元和集成测试报告；⑤确定测试人员限制和时间限制，制定初步测试策略和测试计划，确定测试结束标准。

结果：①建立错误模型，指导回答该软件可能的错误会出现在哪里(用户环境与测试环境不一致会不会出问题，没有测试过的代码或功能里面会不会出问题，没有测试过的输入组合、极端环境或功能使用方法、使用顺序会不会出问题)，我们如何做才能发现这些错误等问题；②在了解测试目标和资源限制之后，按照错误的性价比制定设计测试用例的优先级，并确定初步的测试策略和测试计划；③确定测试结束标准或测试退出机制(已经解决的错误没有重现，所有缺陷报告已经关闭，所有测试用例全部执行完毕，通过错误播种、错误发生曲线分析、历史数据等相应方法统计出所遗留的未发现错误数量可以被接受，以及不属于技术层面且实际表明测试失败或部分失败的市场和管理因素、预算和时间用完等因素)。

2. 设计测试用例

目的：设计能够尽可能多、快、好、省地发现错误的测试用例(即能够找到尽可能多的，以至于所有的bug；能够尽可能快或早地发现最严重的bug；找到的bug是关键的、用户最关心的，且找到bug后能够重现找到的bug，并为修正bug提供尽可能多的信息；能够用最少的时间、人力和资源发现bug，且测试的过程和数据可以重用)。

任务：理解故障模型，理解现有的测试用例库，设计具体的测试用例。

方法：采用基于故障模型(如经验、历史数据/错误、软件开发和运行环境)的软件攻击法(这需要创造性思维，而且要注意保证满足测试用例多、快、好、省的要求，并要有以孙

子兵法进行指导的战役思想，当然还要记住没有银弹的教诲)。

结果：测试用例(IE 4.0 的测试用例数目：10 万)。

测试用例设计的过程是系统测试过程中一个非常重要的环节，它要求从测试的角度对测试计划中的测试需求进行功能和各种特性的细化，确定与被测功能相关的输入/输出变量。继而判断这些变量如何从测试环境中通过硬件接口输入被测软件，以及如何从被测软件的输出中得到。在测试说明中需要最终确定本次测试要测试的系统功能、每一个功能涉及的输入/输出变量以及这些变量取值的等价类划分等。

测试用例没有标准文档格式，对于特殊人或在特殊情况下可以在运行后再形成文档。

测试用例文档由简介和测试用例两部分组成：简介部分描述了测试目的、测试范围、定义术语、参考文档、概述等；测试用例部分逐一列示各测试用例(包含的要素：标题和编号、版本号、修改记录等，针对目标和假设前提/可能发现的错误，输入和数据/代码，测试步骤，预期输出和错误发现方法)。表 11-2 是一个简单的测试用例设计表格。

表 11-2　测试用例设计表格

测试用例 ID	输入			预期结果			实际结果			测试统计		
	利率	贷款期限(年)	贷款金额(元)	月支付	总支付	总利息	月支付	总支付	总利息	通过/失败	测试日期	测试人员
TC-001	8%	30	80000	578.01								
TC-002	8.5%	30	80000	615.13								
TC-003	8.5%	15	80000	787.79								

3. 运行测试用例并处理测试结果

目的：使用测试用例发现错误并关闭错误。

任务：①运行测试用例并记录结果；②评估测试结果并记录缺陷；③处理缺陷直至缺陷关闭(即修改、延迟处理、不修改、不是错误)。

方法：①选择测试用例库中的测试用例运行；②选择新设计的测试用例运行；③录制/回放或笔录中间步骤和结果，记录下执行过程中的灵感(但不要轻易修改本次执行任务)；④分析测试结果并尽量重现和优化错误步骤，详细填写缺陷报告并提供尽可能多的信息(如尽可能提供错误分析和修改建议)，认真审核错误处理结果并及时关闭缺陷报告。

结果：①记录下的运行结果；②记录下的新的测试用例设计思路；③提交并处理的缺陷报告。

4. 评估测试用例和测试设计

目的：检验测试用例和测试设计中测试策略的有效性，必要时对测试用例和测试策略进行完善和修改，增加测试经验。

任务：①根据测试结果完善、修改、合并测试用例，如果没有文档化测试用例，此时需要文档化；②对测试用例库进行维护，即增加新的测试用例(尤其是已经发现了错误的测试用例)，删除不必要的测试用例(要谨慎，除非是功能改变)，修改刚刚使用的测试用例(根

据测试结果)，合并部分测试用例；③根据测试结果完善和修正测试策略和测试计划，产生新的测试用例设计思路。

方法：基于经验(发现了什么问题，这种问题出现的原因是什么，为什么测试用例会发现这种问题，还可以更快地发现吗？测试用例可以合并吗？可能联想到还会出现什么问题——新的测试用例)，流程控制/尤其是测试用例库的维护可以借助工具来实现。

结果：①优化的测试用例库；②优化的软件故障模型；③优化的测试策略和测试计划；④新的测试经验和新的测试用例设计思路。

事实上，测试设计过程是循环往复的，并且过程中的每一步骤都可以返回前面的任何一个步骤，即使单独一个测试用例也可能经历以上步骤多次。另外，测试设计的最重要的工作就是设计测试用例。测试用例的衡量标准：多、快、好、省。测试用例库是一种经验积累，是测试活动最宝贵的财富。最后，在系统测试中设计测试用例的最常用到的思路是软件攻击。

11.1.5　系统测试手段

测试如同打仗，《孙子兵法》中的“知己知彼、百战不殆；攻其薄弱、避其锋芒；分而治之、逐个歼灭”的作战策略在软件测试，特别是软件的系统测试中非常具有指导意义。作战的最主要手段就是进攻或攻击。软件测试中的攻击——软件攻击就是要寻找系统中最容易出现错误的地方进行测试(如寻找开发过程中容易出现疏忽的地方)，保证多、快、好、省地找出错误。软件攻击的核心是基于故障模型的测试用例设计。

1. 软件故障模型

故障模型是将测试员的经验和直觉尽量归纳和固化，使得可以重复使用。测试员通过理解软件在做什么，猜测可能出错的地方，并应用故障模型有目的地使它暴露错误。可以认为故障模型类似于系统设计中的定式(Pattern)思想，具体的攻击方法就是一个一个的定式。因此，对于测试员来说，是能够构造出一个准确的故障模型，使用该故障模型来决定测试策略、测试设计和测试用例。

在建立故障模型时，希望故障模型在框架上是通用的，但是建立具体的故障模型时一定要针对具体的软件类型、应用环境甚至开发工具才有意义。

在进行故障模型设计时，要重点考虑软件的行为特性，如软件功能和技术特点(包括输入、输出、数据以及处理等)、软件操作环境(用户界面、文件系统、操作系统环境以及其他软件及操作系统 API 等)。按照这种思路设计的故障模型是一个二维模型，该模型从软件功能和技术特点来说只接收正确的输入并正确地处理，只输出用户接受的并且正确的输出，保持数据结构的完整性(数值、精度和位置)和自我保护的合法计算、功能交互和数据共享；而从软件操作环境来说主要关注用户界面，关注文件系统接口、数据库系统接口，关注资源调配和管理，以及关注系统 API 调用。

2. 典型攻击方法

基于上述软件行为特性的分析和故障模型设计的考虑，我们重点关注一些典型的攻击方法。这些攻击方法大致从 4 个方面进行考虑：①何时施加攻击，即软件在实现什么功能

的时候适合使用这种攻击，这种攻击针对的功能是什么；②什么样的软件故障会使攻击成功，即本攻击方法主要会暴露实现过程中哪方面的问题，在实现技术上是什么原因产生了这种错误；③如何确定攻击暴露了缺陷，即本攻击方法成功的标志是什么？需要掌握业务背景知识，或者说理解什么是预期的输出结果；④如何进行攻击，即本攻击方法的操作步骤。

下面以 25 种典型攻击方法为例来进行攻击方法的说明。

1) 用户接口输入攻击：6 种

(1) 使用非法输入：所有非法输入有错误处理代码吗？代码正确吗？

(2) 攻击使用默认值作为输入：变量初始化了吗？

(3) 使用特殊字符集和数据类型的合法输入：正确处理特殊字符和数据类型了吗？

(4) 使用使缓冲区溢出的合法输入：检查字符串/缓冲区的边界了吗？

(5) 使用可能产生错误的合法输入组合：输入之间的组合关系考虑了吗？

(6) 重复输入相同的合法输入序列：循环处理的边界考虑到了吗？

2) 用户接口输出攻击：4 种

(1) 产生同一个输入的各种可能输出：一个正确的输入在不同情况下产生不同的输出，这些不同情况都考虑充分了吗？

(2) 强制产生不符合业务背景知识的无效输出：开发人员具有解域/业务背景知识吗？是否会产生不符合业务背景的输出？

(3) 强制通过输出修改一些属性：初始化代码和修改代码同步吗？

(4) 检查屏幕刷新：屏幕刷新时机正确吗？屏幕刷新区域计算正确吗？

3) 用户接口数据攻击：3 种

(1) 制造使内部数据与输入的组合不相容的情况：处理输入的时候，考虑到内部数据的各种可能的组合了吗？

(2) 制造使已有内部数据结构集合溢出的情况：上溢——增加一个元素到集合中，下溢——删除集合中的最后一个元素，或者从空集合中删除元素。

(3) 制造使已有内部数据结构不符合约束的情况：初始化代码和修改代码同步吗？

4) 用户接口计算攻击：4 种

(1) 使用非法的操作数和操作符组合，攻击用户可以控制计算要求的情况：考虑到操作符和计算要求的合法性了吗？

(2) 使函数递归调用自身：考虑到循环/递归的中止了吗？

(3) 使计算结果溢出：考虑数据结构是否能够正确存储可能的计算结果了吗？

(4) 攻击共享数据或互相依赖的计算功能：一个函数在修改共享数据的时候考虑到其他函数对共享数据的假设/约束了吗？

5) 文件系统介质攻击：3 种

(1) 使文件系统超载：检查文件访问函数的返回值了吗？正确处理文件访问失败的情况了吗？

(2) 使介质处于忙或不可用状态：检查文件/介质访问函数的返回值了吗？正确处理文

件/介质访问失败的情况了吗？

(3) 损坏介质(这时候 OS 可能认为介质可用)：检查文件/介质访问函数的返回值了吗？正确处理文件/介质访问失败的情况了吗？

6) 文件系统文件攻击：3 种

(1) 使用特殊字符/特殊长度/无效的文件名：处理文件名的代码考虑到各种情况了吗？

(2) 改变文件访问权限：访问文件的函数考虑到文件访问权限了吗？文件访问失败，有正确处理错误的代码吗？

(3) 使文件内容错误，并让系统使用这个文件：检查文件访问函数的返回值了吗？错误处理代码正确吗？

7) 操作系统和软件接口攻击：两种

(1) 记录-仿真攻击：模拟操作系统和操作环境故障，并记录软件对该故障的反应。例如，软件正确处理内存故障/网络故障等操作系统和软件接口故障了吗？

(2) 观察-失效攻击：观察底层 API 调用，并动态修改 API 调用，制造错误。例如，软件正确处理操作系统和软件 API 调用错误的情况了吗？该攻击方法一般用于对可靠性和稳定性要求非常高的软件。

上述攻击中的许多要用到软件攻击工具，如文件系统介质攻击、文件系统文件攻击、操作系统和软件接口攻击等。否则：①攻击成本太高，比如物理毁坏介质；②工作量太大，比如使用大文件填充硬盘、使用多任务抢占 CPU/网络/内存等资源；③有一些不容易实现，比如动态监控和修改 API 调用。我们一般采用软件故障植入的方法设计软件攻击工具。

3. 软件故障植入

一般软件中的程序代码分为功能代码(通过实现用户需求来完成软件任务)和异常处理代码(通过异常或其他错误处理机制来处理程序错误)，故障植入的目标是强制执行异常处理代码(没有异常处理代码也是一种错误)，从而发现其中的错误。

故障植入方法分为编译期植入(在源代码中插入引发故障现象的代码)和运行期植入(在目标代码或运行环境中植入引发故障现象的代码)两类。其中，运行期软件故障植入具有更大的优势：①不需要源代码，因为系统测试阶段经常没有源代码；②可以达到模拟环境故障的目标，比如网络中断、网络繁忙、内存匮乏等；③可以以 API 调用失败的方式模拟故障，因为在程序看来，任何环境故障实质上都是一系列的 API 调用失败；④可以只影响被植入的程序，因为使用的是模拟 API 调用失败方式。

运行期软件故障植入可通过截获 API 的方式来实现。一般用到三种方式：①基于调用源截获，找出程序中调用 API 的名字或 DLL，然后用自己的函数替换掉；②路径内截获，如果程序中使用 VTable 等技术间接记录调用 API 地址，则直接修改 VTable 中的 API 地址，用自己的函数替换掉；③目的地截获，修改被调用 API 地址，修改头部代码，转向自己的函数，这实际上采用的是病毒方式。

运行期软件故障植入的实现策略有两种：

(1) 基于模式的故障植入：模拟环境故障，记录故障特征和影响到的 API，修改影响到的 API；采用乱棒打师傅方式的软件攻击法——canned HEAT(Hostile Enviroment

Application Tester)，并记录由此引起的故障产生情况。

(2) 基于系统调用的故障植入：观察使用的 API，并独立地、细粒度地修改影响任何一个 API；采用偷梁换柱、李代桃僵方式的软件攻击法，并观察因其破坏性而引起的失效攻击。

4. 软件攻击的突破口

对系统质量影响最大的地方是系统最薄弱、最容易出问题的地方，这些地方也是软件攻击的突破口。

1) 错误处理测试

健壮性是软件质量的一个重要因素。错误处理测试是检查软件在面对错误时，是否进行了正确的处理。

错误处理测试的目的是要发现软件是否做了用户不期望的事情，发现软件在发生异常的时候是否有能力进行处理。此时，测试人员需要以否定的态度来思考问题。另外，在错误处理测试中发现的部分问题可能不会被修复。

错误处理测试主要考虑典型的异常情况，如用户输入非法数据(不输入数据；输入无效数字数据，如负数和字母数字串；输入任何被认为是非法的数据类型格式；尝试不常用的数据组合；确保使用零值；输入超过或短于要求长度的数据)，在系统不支持的平台上运行，网络连接异常，数据文件(或数据库)被破坏，数据文件(或数据库)中有混乱的数据，计算机断电后启动，用户界面上的违反操作步骤的操作等。

2) 内存泄漏测试

内存泄漏是一种典型的程序缺陷，导致应用程序不断消耗系统内存(或虚拟存储器)，使程序运行出现响应变慢、某些功能无法实现甚至整个系统瘫痪等问题。尤其对于嵌入式系统这种资源比较匮乏、应用非常广泛且往往又处于重要部位的程序，内存泄漏将可能导致无法预料的重大损失。在用某些语言(如 C/C++语言)编写的程序中，内存泄漏是一个极其普遍的问题。

内存泄漏测试可采用静态测试和动态测试技术。首先，通过测量内存使用情况，了解程序内存分配的真实情况，发现对内存不正常的使用；另外，在问题出现前发现征兆，在系统崩溃前发现内存泄漏错误；最后，发现内存分配错误，并精确显示发生错误时的上下文情况，指出发生错误的原因。

3) 性能测试

- 性能测试包含并发性能测试、强度测试、破坏性测试等方面。
- 并发性能测试是评估系统交易或业务在渐增式并发情况下处理瓶颈以及能够接收业务的性能过程。
- 强度测试是在资源受限的情况下，找出因资源不足或资源争用而导致的错误。
- 破坏性测试重点关注超出系统正常负荷若干倍的情况下，错误出现状态和出现比率以及错误的恢复能力。

性能测试可以通过黑盒测试或白盒测试方法来进行，一般要借助工具，如 HP LoadRunner 等。

在进行性能测试时，要求：①测试程序在获得定量结果时程序计算的精确性；②测试程序在有速度要求时完成功能的时间；③测试程序完成功能所能处理的数据量；④测试程序各部分的协调性，如高速、低速操作的协调性；⑤测试软/硬件中哪些因素限制了程序的性能；⑥测试程序的负载潜力；⑦测试程序运行占用空间。

性能测试应用场合有：①软件中某个模块涉及复杂的计算，特别是一些基于人工智能的分析；②涉及大量数据的读写、通信；③涉及数据检索，而被检索的数据具有很大的数据量；④具有多个并发用户；⑤软件在运行时，可用资源(特别是 CPU 和内存)可能在某些情况下很紧张，例如一些嵌入式系统软件。

4) 用户界面测试

用户界面测试和评估的重点是正确性、易用性和视觉效果，界面中的文字检查和拼写检查也是用户界面测试的重要环节。在用户界面测试的过程中，有时要依赖于测试人员的主观判断，但用户界面测试也要遵循一些基本原则，如易用性、规范性、合理性、美观与协调性、菜单位置、独特性、快捷方式的组合、排错性考虑等。表 11-3 给出了界面测试的一些基本考虑。

表 11-3　界面测试指标

指标	检查项	测试人员评价
合适性和正确性	用户界面是否与软件的功能相融洽？	
	是否所有界面元素的文字和状态都正确无误？	
容易理解	对于常用的功能，用户能否不必阅读手册就能使用？	
	是否所有界面元素(例如图标)都不会让人误解？	
	是否所有界面元素提供了充分而必要的提示？	
	界面结构能否清晰地反映工作流程？	
	用户是否容易知道自己在界面中的位置，不会迷失方向？	
	有联机帮助吗？	
及时反馈信息	是否提供进度条、动画等反映正在进行的比较耗时间的过程？	
	是否为重要的操作返回必要的结果信息？	
出错处理	是否对重要的输入数据进行校验？	
	执行有风险的操作时，有确认、放弃等提示吗？	
	是否根据用户的权限自动屏蔽某些功能？	
	是否提供 Undo 功能用以撤销不期望的操作？	
风格一致	同类的界面元素是否有相同的视感和操作方式？	
	字体是否一致？	
	是否符合广大用户使用同类软件的习惯？	
适应各种水平的用户	所有界面元素都具备充分必要的键盘操作和鼠标操作吗？	
	初学者和专家都有合适的方式操作这个界面吗？	
	色盲或色弱的用户能正常使用该界面吗？	
国际化	是否使用国际通行的图标和语言？	
	度量单位、日期格式、人的名字等是否符合国际惯例？	

(续表)

指标	检查项	测试人员评价
布局合理、色彩和谐	界面的布局符合软件的功能逻辑吗？	
	界面元素是否在水平或垂直方向对齐？	
	界面元素的尺寸是否合理？行、列的间距是否保持一致？	
	是否恰当地利用窗体和空间的空白，以及分割线条？	
	窗口切换、移动、改变大小时，界面正常吗？	
	界面的色调是否让人感到和谐、满意？	
	重要的对象是否用醒目的色彩表示？	
	色彩使用是否符合行业的习惯？	
个性化	是否具有与众不同的、让用户记忆深刻的界面设计？	
	是否在具备必要的一致性的前提下突出个性化设计？	

5) 压力测试

压力测试也叫负荷测试，即获取系统能正常运行的极限状态。压力测试用于检查软件在面对大数据量时是否可以正常运行大数据量，往往是发生概率比较小的情况。

压力测试所涉及的方面主要包括数据库大小、磁盘空间、可用内存空间、数据通信量。表 11-4 给出了压力测试的测试模板。

表 11-4　压力测试模板

极限名称	如最大并发用户数量	
前提条件		
输入/动作	输 入	是否能正常运行
如 10 个用户并发操作		
如 100 个用户并发操作		

6) 软件运行时错误

软件运行时错误是指应用程序在运行期间执行了非法操作或某些操作失败时出现的错误，是所有软件错误中最具风险的。下面的图 11-3~图 11-5 说明了软件运行时错误产生的例子。



图 11-3　Windows 2000 操作系统上发生软件运行时错误的现象

在这种情况下，不管我们做什么选择，应用程序都会退出。可能对于一般的软件来说，出现这样的错误没关系，但对于航空航天、汽车以及医疗设备等安全级别要求非常高的系统来说，一旦出现这样的运行错误，损失就是不可估量的。因此，对于关键或重要的软件必须进行运行时检查。

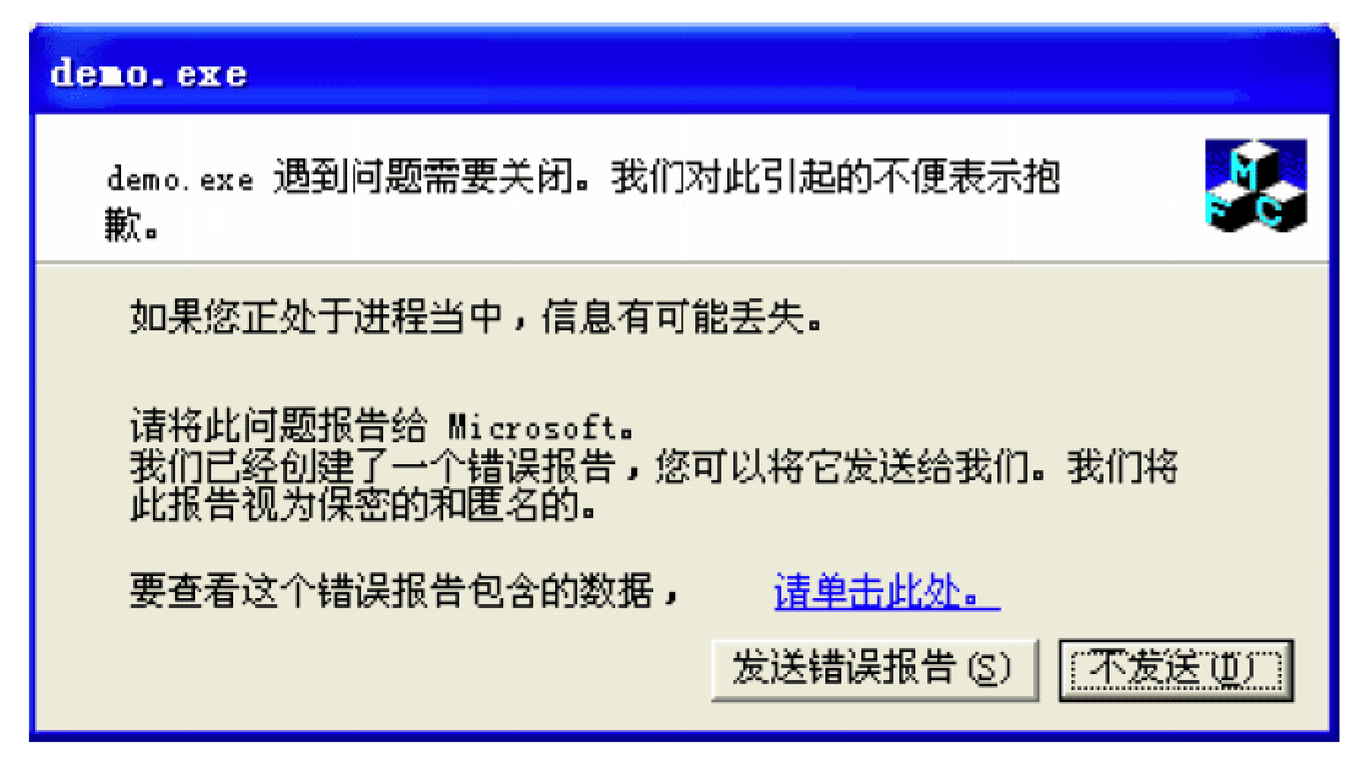


图 11-4　Windows XP 操作系统上发生软件运行时错误的现象

```
$ gcc unixRTE.c
unixRTE.c:9:2: warning: no newline at end of file

Administrator@LYH /cygdrive/e/polyspace_Training
$ ./a.exe
0
Arithmetic exception (core dumped)

Administrator@LYH /cygdrive/e/polyspace_Training
$
```

图 11-5　UNIX 操作系统上发生软件运行时错误的现象

运行时检查是实际运行时检测程序的方案，通过选择适当的测试用例，并对程序的运行状态进行监控以发现程序中的错误。该技术依靠系统编译程序和动态检查工具实现检测，优点是检测的结果比较接近于程序的真实运行状态，对于内存使用情况、空指针引用等错误的检测比较准确。但缺点是检测的全面性严重依赖于测试用例对代码的覆盖情况，运行时间较长，而且对于程序中的递归调用等过程，动态检测很难覆盖全部情况。

6) 回归测试

回归测试是指对某些已经被测试过的内容进行重新测试，如软件增加后影响软件的结构，软件修改考虑不周而引入问题。

回归测试的目标有两个：①检查测试出的软件问题是否得到了正确修改；②保证被测软件在被修改之后，各项功能依然正确(未引入新的缺陷)。

一般来说，应该对修改的部分进行针对性的测试以保证其符合需求，随后还要运行一系列的测试以确认现有功能仍然符合需求。

在工程实践中使用的回归测试有多种策略：①通过进行用例的相关性分析，找到与软件修改部分相关的测试用例进行测试；②在不进行任何分析的前提下对测试用例的全集进行测试；③必要时可能会生成新的测试用例，专门用来进行回归测试；④测试人员可以按照实际情况选择回归策略，如每两周需要进行一次完整的回归测试。或当修复的缺陷数量累计 50 个时，进行一次完整的回归测试。也可以在产品递交用户前 5 个工作日，进行完整的回归测试；⑤回归测试通常要借助自动化测试工具。

11.2　系统测试工具

自动化测试需要不同类型的自动化测试工具进行支持。现在市面上有商用软件测试工具，包括 IBM Rational Robot、HP UFT 等，还有一些开放源代码的测试工具，如 Selenium、JMeter、Ant、JUnit、JProbe 和 Cactus 等。

11.2.1　功能自动化测试工具 Selenium 及其应用

Selenium 是 ThoughtWorks 公司用于 Web 应用程序测试的工具，通过模拟用户对 Web 页面的各种操作，可以精确重现软件测试人员编写的 Test Cases 步骤。Selenium 测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括 IE、Mozilla Firefox、Mozilla Suite 等。其他测试工具都不能覆盖如此多的平台。

使用 Selenium 和在浏览器中运行测试还有很多其他好处。下面是主要的两大好处：

(1) 通过编写模仿用户操作的 Selenium 测试脚本，可以从终端用户的角度来测试应用程序。

(2) 通过在不同浏览器中运行测试，更容易发现浏览器的不兼容性。

Selenium 的核心，也称 browser bot，是用 JavaScript 编写的。这使得测试脚本可以在受支持的浏览器中运行。browser bot 负责执行从测试脚本接收到的命令，测试脚本要么是用 HTML 的表格布局编写的，要么是使用一种受支持的编程语言编写的。

Selenium 的主要功能包括：测试与浏览器的兼容性——测试应用程序是否能够很好地工作在不同浏览器和操作系统之上。测试系统功能——检验软件功能和用户需求。Selenium 支持自动录制操作和自动生成.NET、Java、Perl 等不同语言的测试脚本。Selenium 是 ThoughtWorks 专门为 Web 应用程序编写的一个可进行集成测试、系统测试及验收测试的测试工具。

Selenium 包含三个工具：Selenium-IDE、Selenium-RC 以及 Selenium-Core。其中，Selenium-Core是驱动Selenium工作的核心部分，作为一个用JavaScript编写的测试引擎，它可以操作Web页面上的各种元素，诸如单击按钮、输入文本框以及断言Web页面上存在某些文本与Web元素等，支持Windows平台上的IE、Firefox、Chrome等浏览器以及Linux平台。

Selenium-IDE 是一个 Firefox 插件，能够录制、回放用户在 Firefox 中的行为，并把所记录的 Selenese(Selenium Commands)转为 Java/C#/Python/Ruby 等语言编码，在 Selenium-RC 中修改重用。对于较为复杂的 Test Cases，Selenium-IDE 的功能有限，往往用它录制大致的步骤，再转换为测试人员熟悉的编程语言，在此基础上完善，形成更为强大且灵活的 Selenium-RC Test Cases。

Selenium-RC(Selenium Remote Control)在 Web 浏览器与需要测试的 Web 应用间架设代理服务器(Selenium Server)，使得 JavaScript 引擎与被测 Web 应用同源，绕开同源策略的限制(Same Origin Policy)，进而取得对 Web 页面进行各种操作的权限。

1. Selenium 环境的建立

这里选用的是最新版本的 Selenium 组件——Selenium 2.0.0 版本，与其匹配的火狐浏览

器为 Firefox 19.0 或 Firefox 20.0。

1) 安装 Firefox

下载火狐浏览器 Firefox 20.0，下载地址为http://firefox.com.cn/download/，下载后安装到系统默认路径 C:\Program Files\Mozilla Firefox 下。

2) 安装 Selenium

下载 Selenium IDE，下载地址为http://docs.seleniumhq.org/download/，下载后将 xpi 文件复制到 Firefox 安装目录的 extensions 子目录下，路径为 C:\Program Files\Mozilla Firefox\extensions，然后重启 Firefox。在菜单栏的“工具”菜单中显示 Selenium IDE，单击该菜单项，弹出 Selenium IDE 2.0.0 对话框，如图 11-6 所示，说明安装成功。

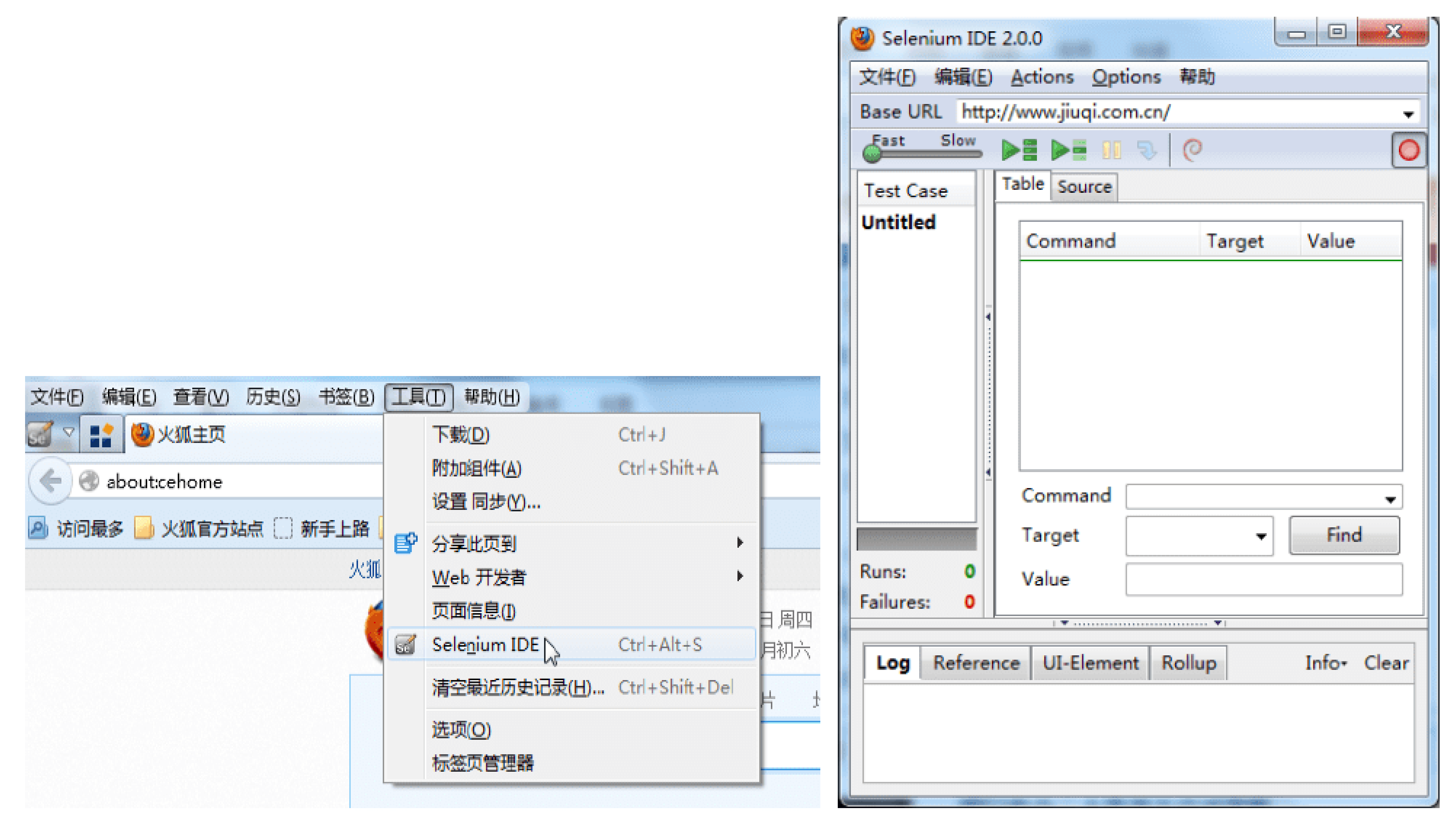


图 11-6　Selenium 安装成功

3) 启动测试服务

启动久其软件官方网站服务，在 Firefox 浏览器中输入 http://www.jiuqi.com.cn/，显示图 11-7 所示页面，证明服务启动成功。



图 11-7　服务启动成功

2. 应用流程

1) 新建测试集

测试集是一组测试用例，是 Selenium IDE 可以播放的最大单元。新建测试集非常简单，在 Selenium IDE 的“文件”菜单中单击 New Test Suite 命令即可，如图 11-8 所示。

2) 新建测试用例

测试用例是 Selenium 管理的最小单元，一个测试集可以有多个测试用例，测试用例可以单个播放，也可以套件播放。新建测试用例也非常简单，只需要单击 Selenium IDE 菜单中的 New Test Case 命令即可，如图 11-9 所示。

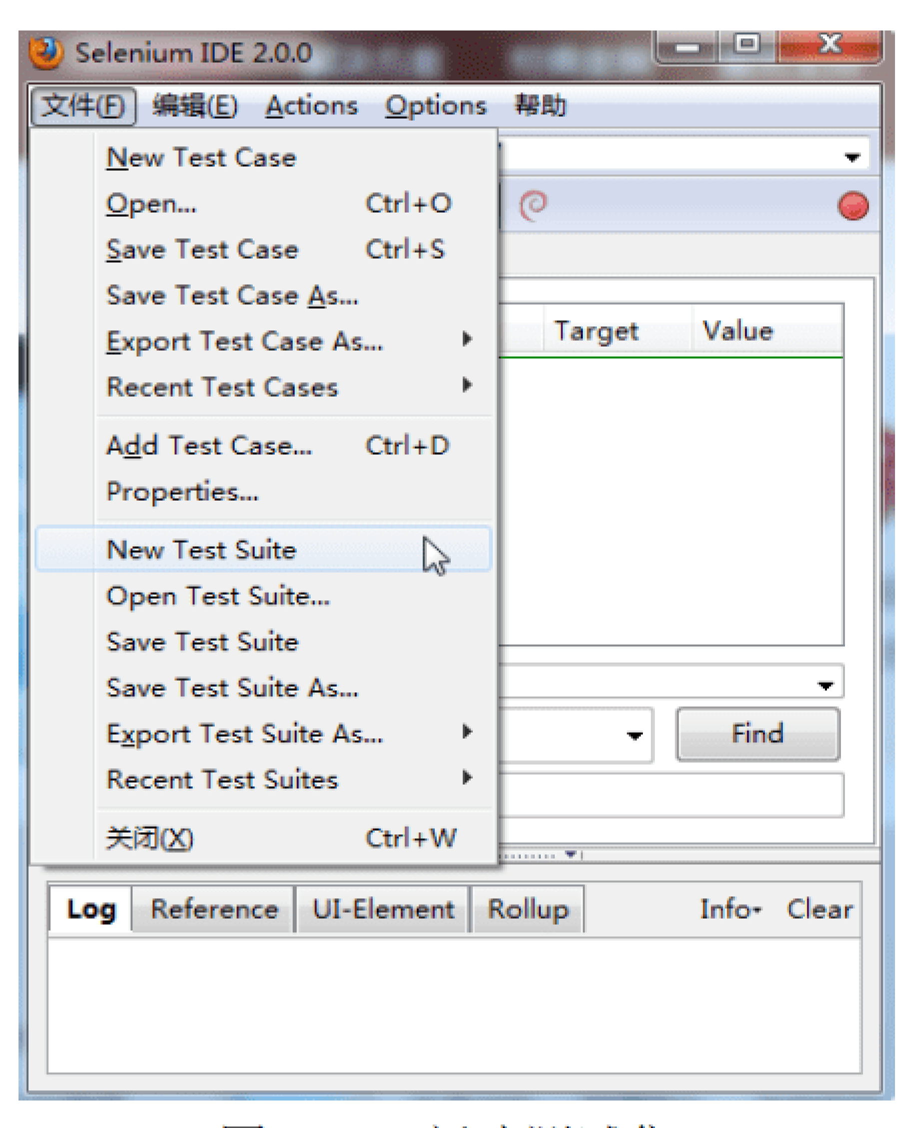


图 11-8　新建测试集

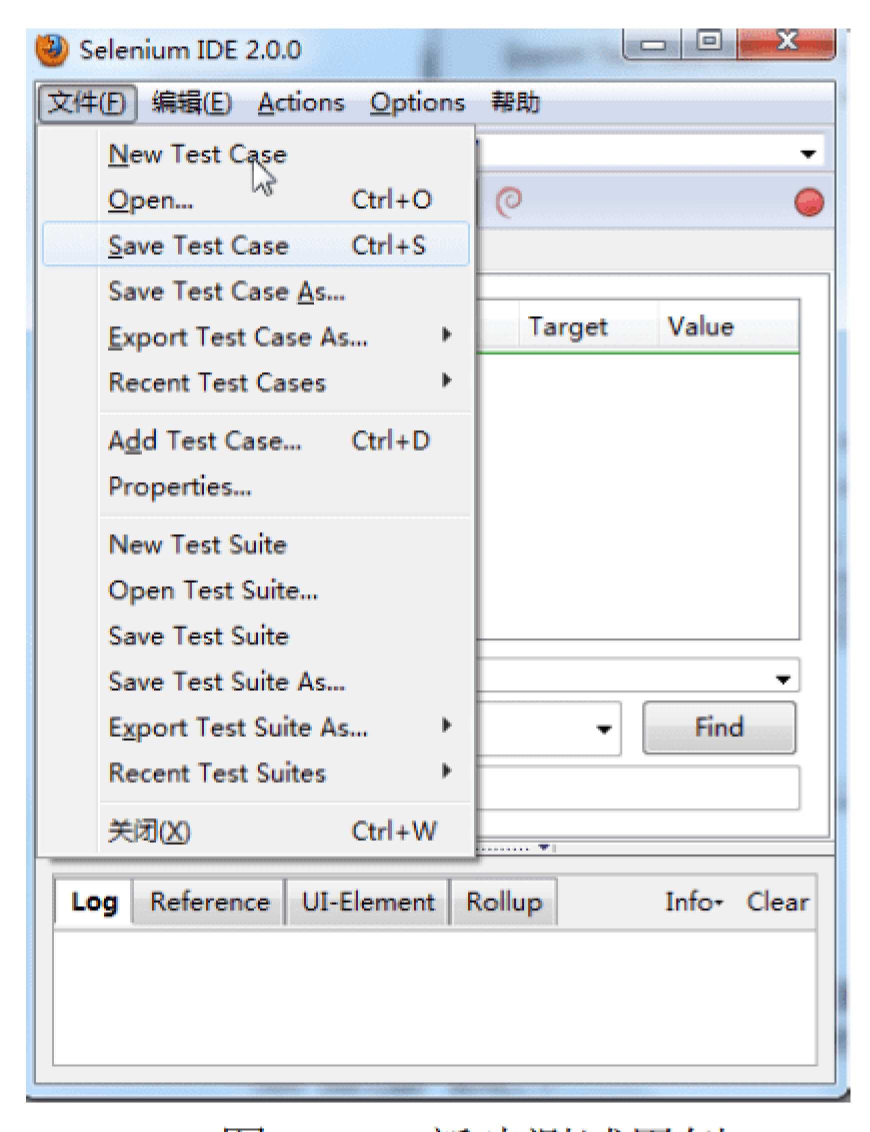


图 11-9　新建测试用例

3) 录制测试脚本

创建了测试用例后，就可以录制用例脚本，单击 IDE 界面中的红色按钮，启动脚本录制，如图 11-10 所示。

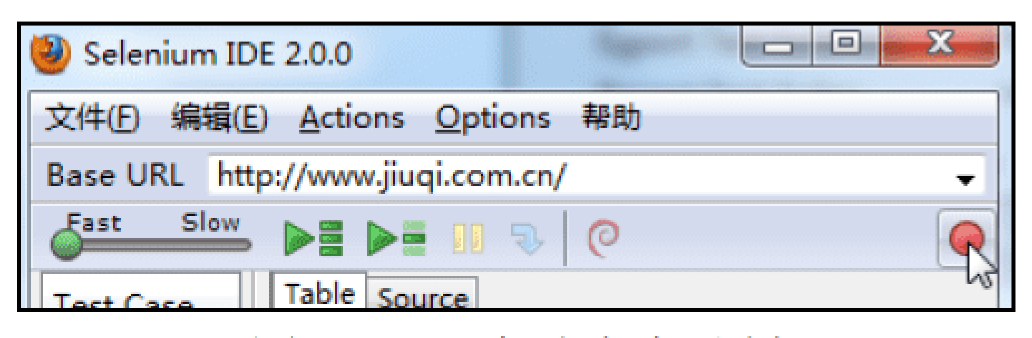


图 11-10　启动脚本录制

然后在系统界面中执行相应的操作，直到用例完成，再次单击红色按钮，结束脚本的录制，IDE 界面中将显示用例对应的脚本内容，例如 11-11 图中的 Table 选项卡，其中包含 Command、Target、Value 等信息。

4) 保存测试用例

测试用例脚本录制完毕后，可以对脚本进行保存，只需要单击 Selenium IDE 的“文件”菜单中的 Save Test Case 命令，然后提供以下测试用例的名称，比如测试用例 01，单击“确定”即可，指定的目录中会多一个.html 文件。保存后 IDE 界面左侧显示了保存后的测试用例名称，选中后右侧界面会显示对应的脚本信息，如图 11-12 所示。

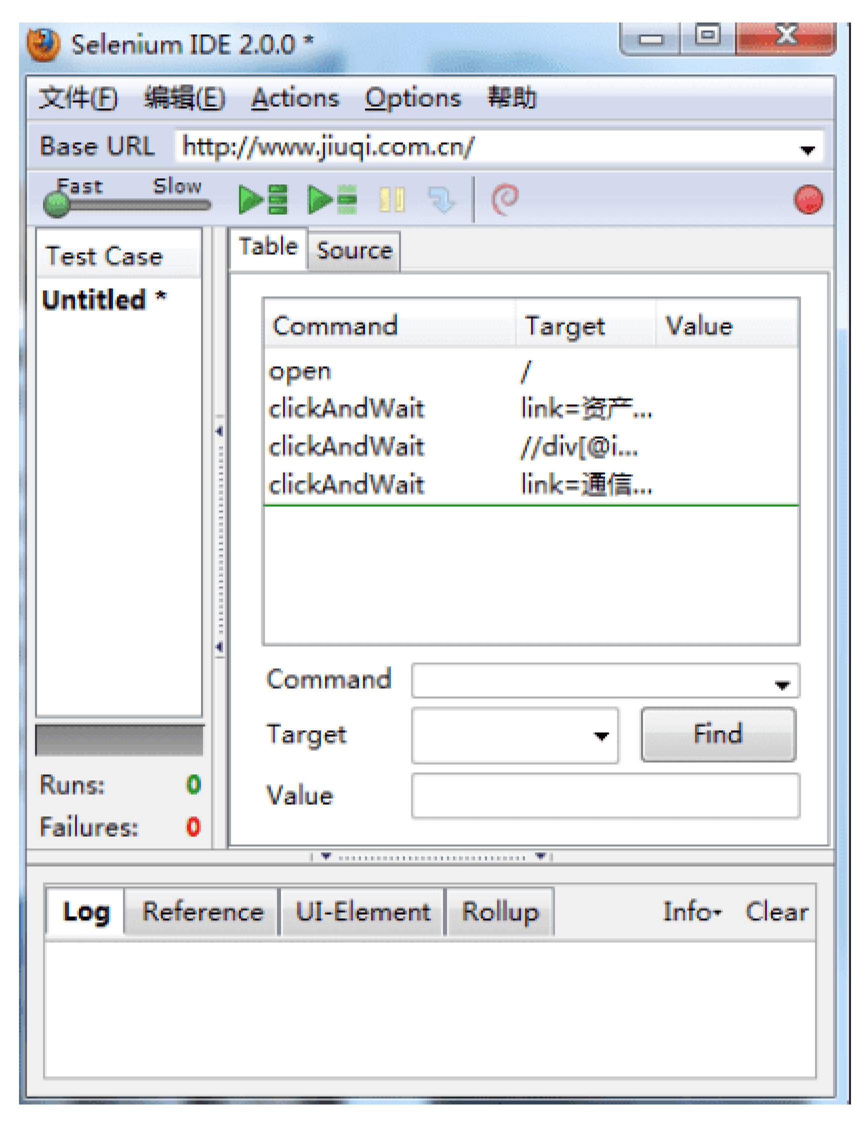


图 11-11　结束脚本录制的情况

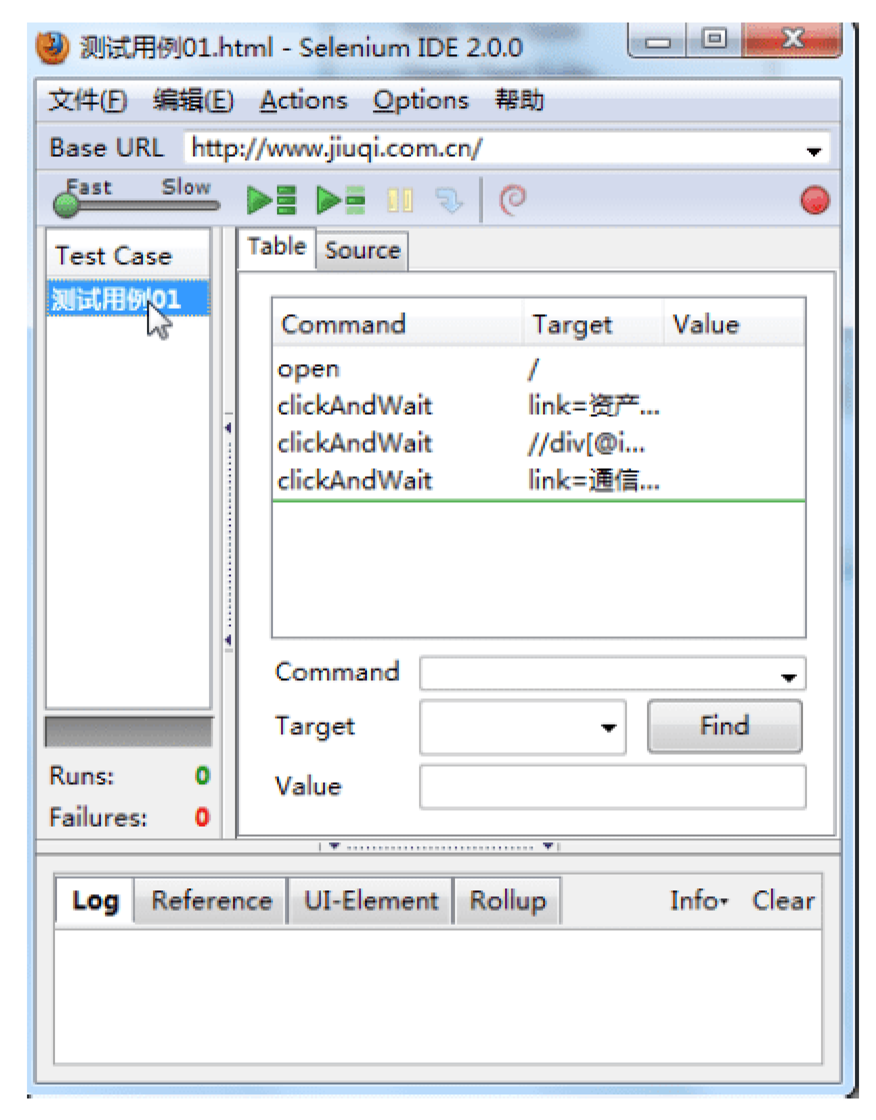


图 11-12　测试用例的保存情况

5) 保存测试集

保存了多个测试用例后，可以将这些测试用例保存到一个测试集中，只需要单击“文件”菜单中的 Save Test Suite 命令，提供测试集的名称，比如测试集 01，单击“确定”即可，指定的目录中会多出一个没有后缀名的文件。

6) 打开测试集

保存后的测试集可以被IDE再次打开，单击“文件”菜单中的Open Test Suite命令，选择测试集所在的目录，选中后单击“确定”即可。打开后IDE界面中将显示该测试集的用例信息，如图11-13所示。

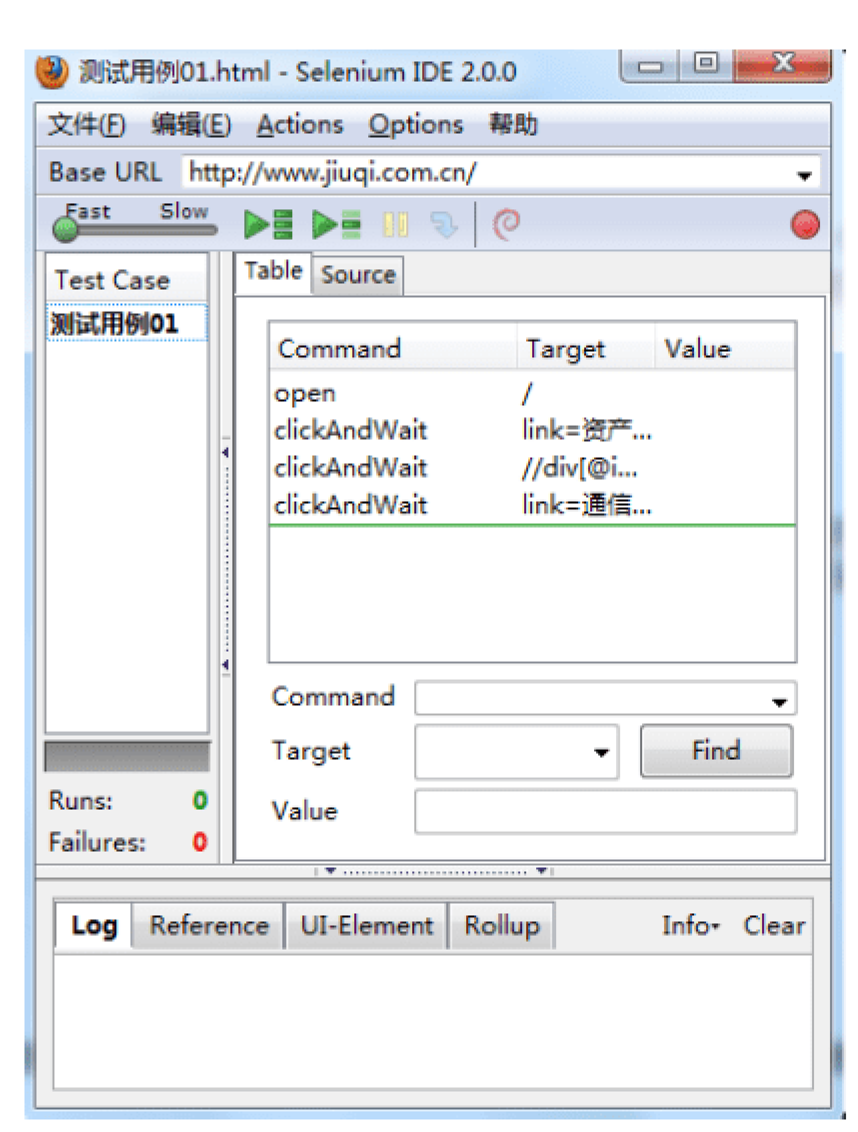


图 11-13　测试集的打开

7) 回放测试用例

录制的测试用例脚本可用来修改、补充和回放。在 IDE 中准备好基本的脚本后，确定输入内容正确无误，做好验证设定，可以回放当前脚本，最终 IDE 会给出提示通过情况和不通过情况。

在录制完成之后回放，经常会遇到回放失败的过程，所以需要对 IDE 的脚本录制进行修改。录制回放中的经验主要涉及以下几点：

(1) 定位不准，如果 XPath 定位不准，可自己根据页面输入 XPath，也可以通过 ViewPath 和 FireBug(都是火狐浏览器的插件)来定位 XPath 脚本，即 Target。

(2) click 是最常用的单击事件，如果在回放的过程中不执行 click 事件，可以试试其他的事件，比如 MouseDown 等。

(3) 有时打开了多个页面，IDE 无法定位哪个页面为当前的脚本执行页面，需要使用 SelectWindow 来定位。

IDE 提供了两种回放方式，即回放单个测试用例和回放当前测试集中的所有测试用例，并且可以设置回放的速度，回放完毕后，界面将显示测试用例回放的结果，包括回放测试用例的个数、失败的个数等，如图 11-14~图 11-17 所示。

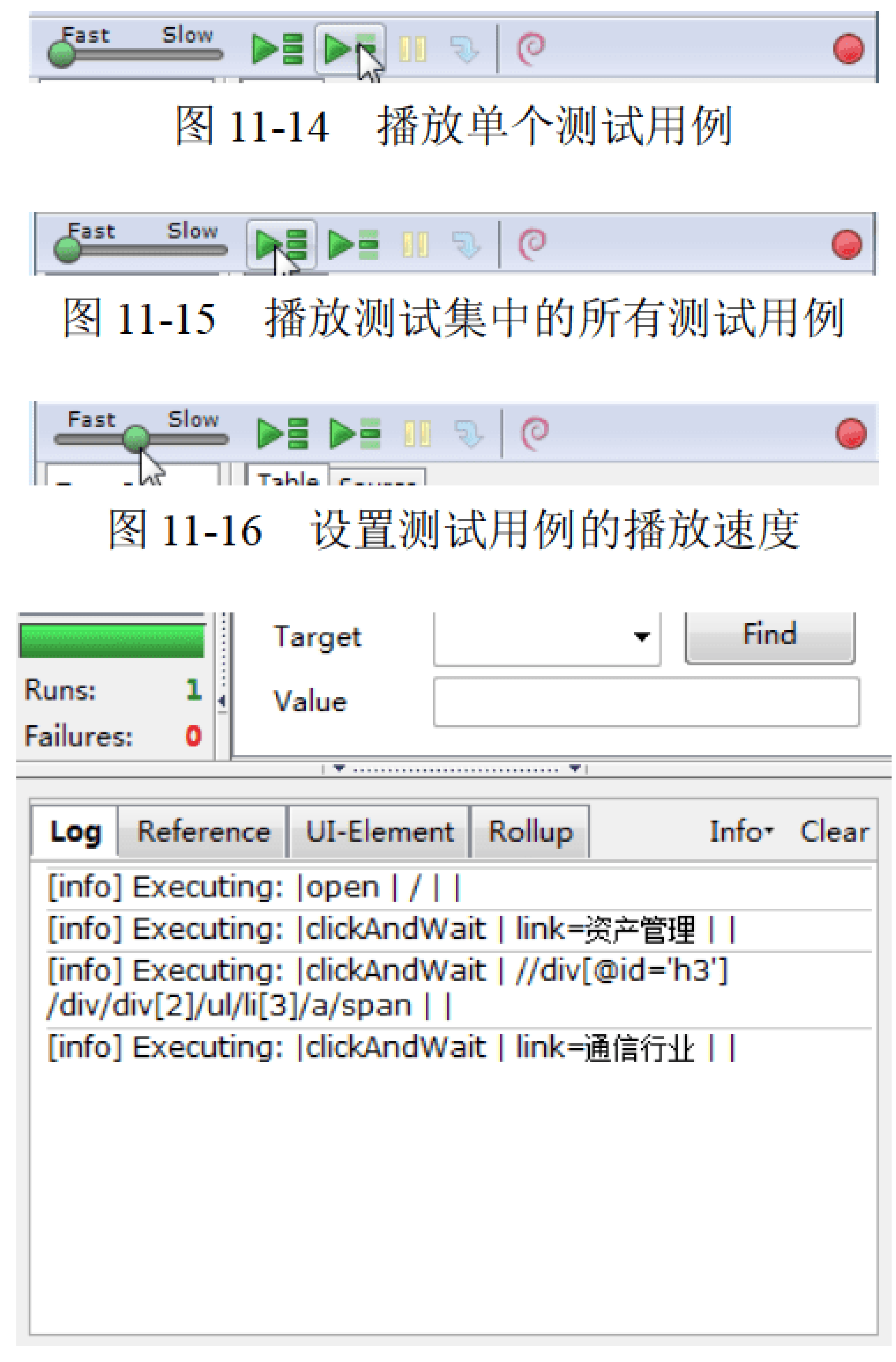


图 11-14　播放单个测试用例

图 11-15　播放测试集中的所有测试用例

图 11-16　设置测试用例的播放速度

图 11-17　显示回放结果

8) 基于 IDE 调试脚本

在 IDE 中调试脚本，如图 11-18 所示

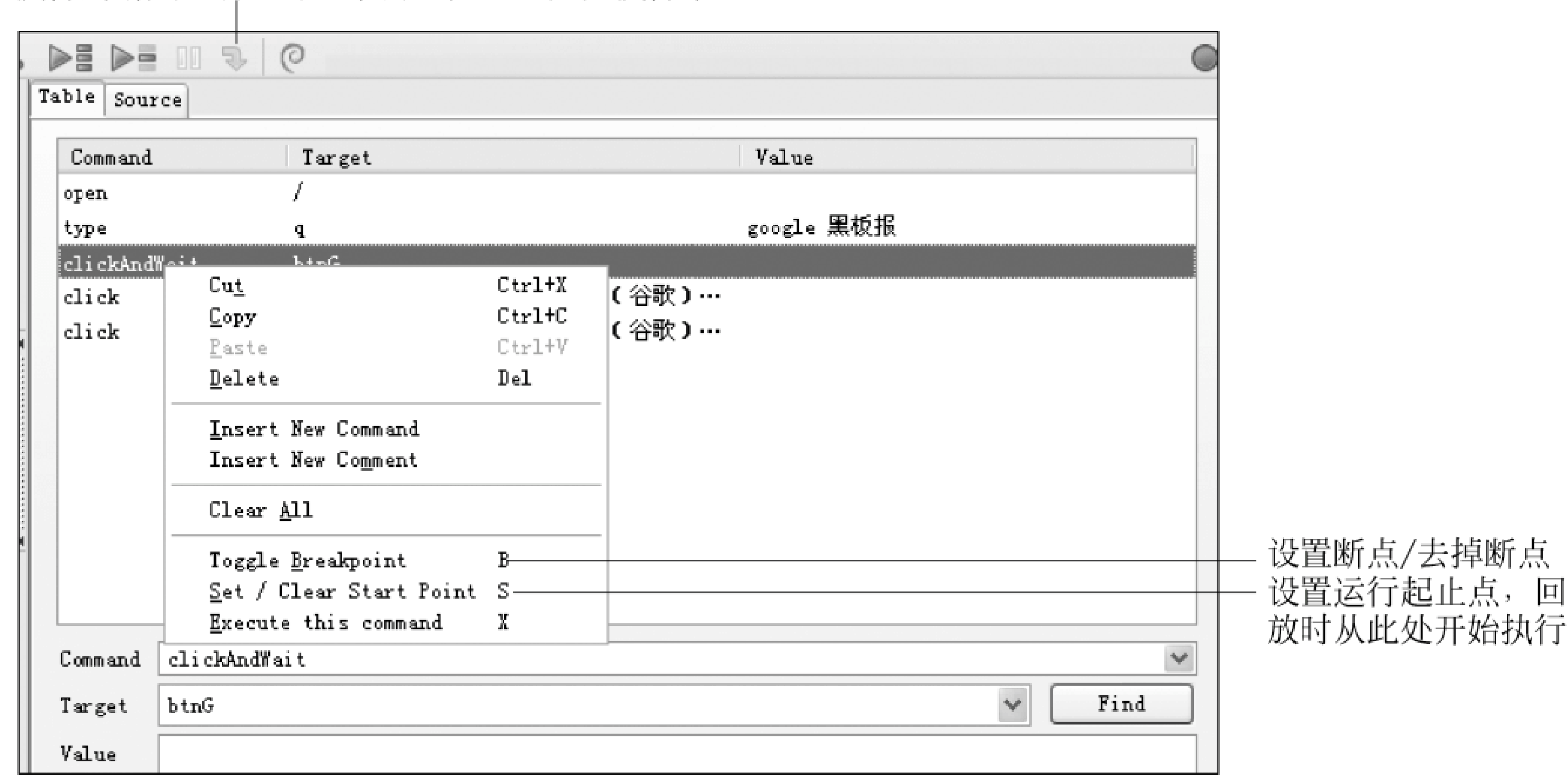


图 11-18　脚本调试

9) 把录制的内容转换成 Java 脚本

将录制的内容转换成 Java 脚本的方法如图 11-19 所示。

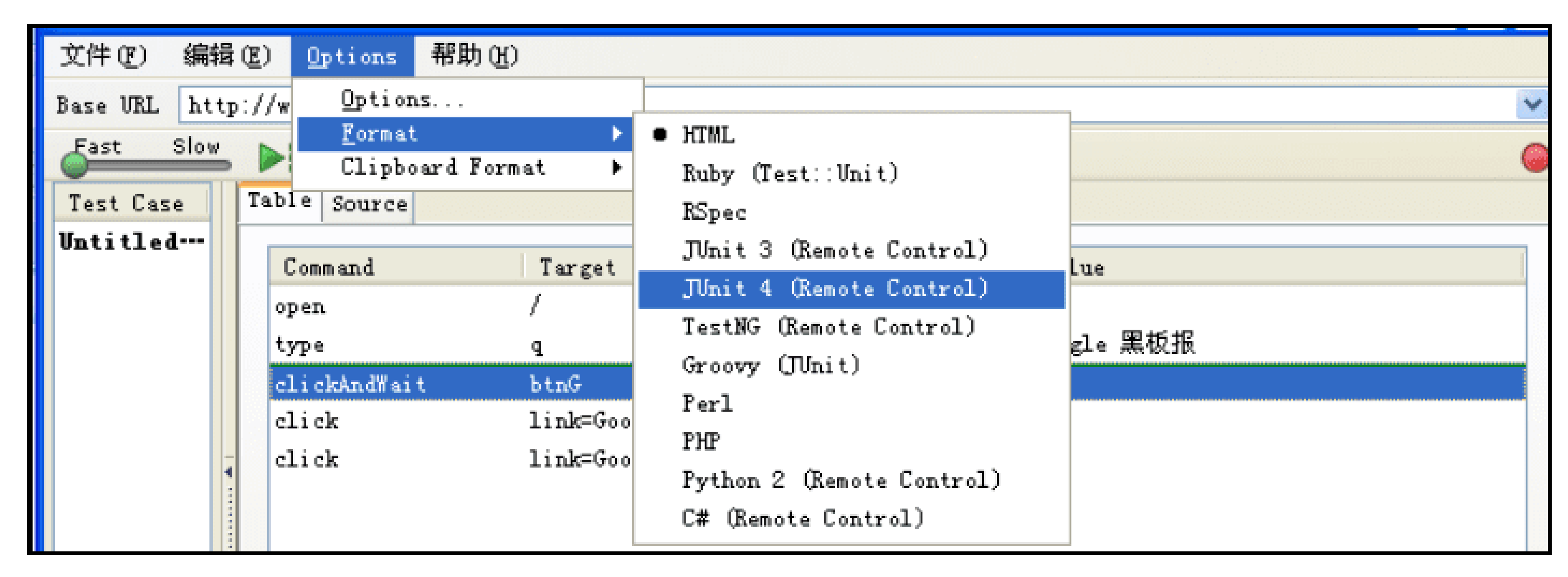


图 11-19　将录制的内容转换成 Java 脚本

3. 应用举例

1) 系统需求

成功登录高校学生选课管理系统后，网站显示功能包括专业管理、课程管理、统计信息、修改密码、退出系统。

2) 脚本录制

(1) 用 Firefox 访问高校学生选课管理系统，进入主界面。

(2) 单击 Firefox 菜单栏中的“工具”| Selenium IDE 命令，打开 Selenium IDE 窗口，单击红色按钮，开始测试脚本录制，如图 11-20 所示。

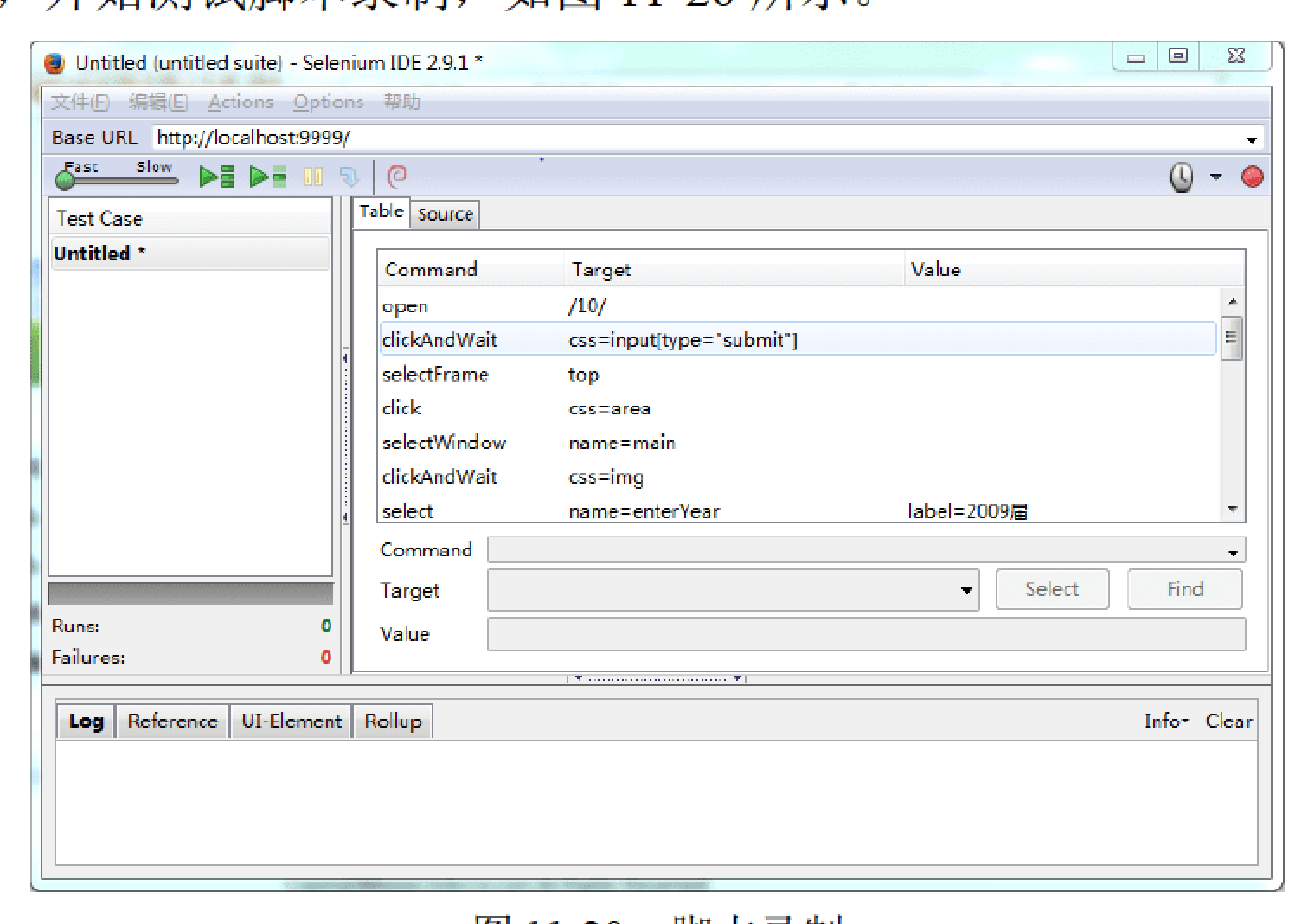


图 11-20　脚本录制

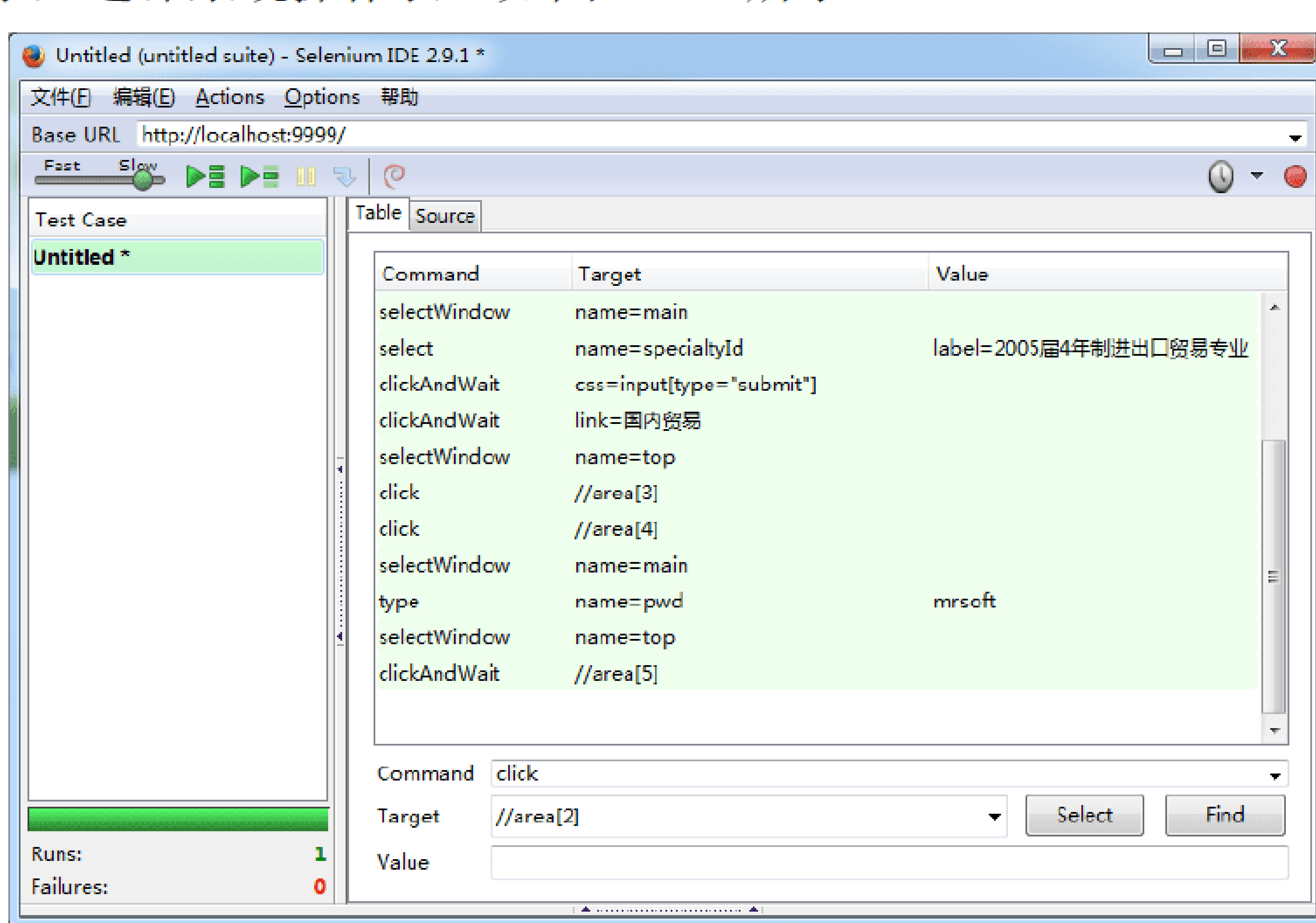
(3) 回到主界面，选择一个专业名称。依次单击“专业管理”、“课程管理”、“统计信息”、“修改密码”、“退出系统”功能模块按钮。

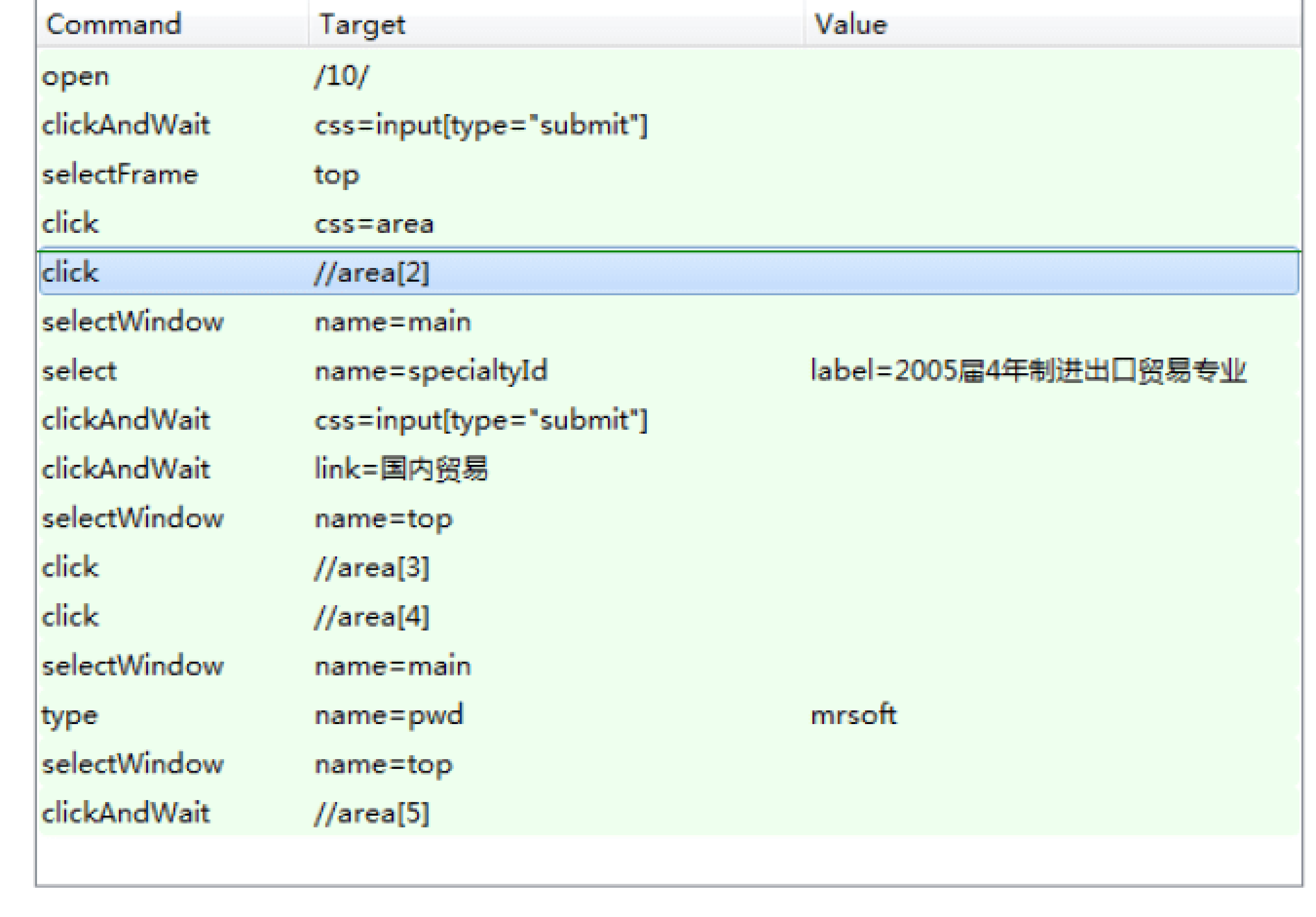
(4) 再次单击红色按钮，结束测试脚本的录制。保存测试用例，用例名为“专业管理”；并保存测试集，将测试集命名为“高校学生选课系统”。

3) 脚本回放

打开测试集“高校学生选课系统”，选中“高校学生选课系统”测试用例，单击“回放测试用例”按钮，执行测试脚本回放，回放完成后，界面输出以下内容：

(1) 高校学生选课系统操作表，如图 11-21 所示。





Command	Target	Value
open	/10/	
clickAndWait	css=input[type="submit"]	
selectFrame	top	
click	css=area	
click	//area[2]	
selectWindow	name=main	
select	name=specialtyId	label=2005届4年制进出口贸易专业
clickAndWait	css=input[type="submit"]	
clickAndWait	link=国内贸易	
selectWindow	name=top	
click	//area[3]	
click	//area[4]	
selectWindow	name=main	
type	name=pwd	mrsoft
selectWindow	name=top	
clickAndWait	//area[5]	

图 11-21　界面操作情况

(2) 操作生成的脚本程序：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="http://localhost:9999/" />
<title>New Test</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">New Test</td></tr>
</thead><tbody>
<tr>
    <td>open</td>
    <td>/10/</td>
```

```
        <td></td>
    </tr>
    <tr>
        <td>clickAndWait</td>
        <td>css=input[type=&quot;submit&quot;]</td>
        <td></td>
    </tr>
    <tr>
        <td>selectFrame</td>
        <td>top</td>
        <td></td>
    </tr>
    <tr>
        <td>click</td>
        <td>css=area</td>
        <td></td>
    </tr>
    <tr>
        <td>click</td>
        <td>//area[2]</td>
        <td></td>
    </tr>
    <tr>
        <td>selectWindow</td>
        <td>name=main</td>
        <td></td>
    </tr>
    <tr>
        <td>select</td>
        <td>name=specialtyId</td>
        <td>label=2005 届四年制进出口贸易专业</td>
    </tr>
    <tr>
        <td>clickAndWait</td>
        <td>css=input[type=&quot;submit&quot;]</td>
        <td></td>
    </tr>
    <tr>
        <td>clickAndWait</td>
        <td>link=国内贸易</td>
        <td></td>
    </tr>
    <tr>
        <td>selectWindow</td>
        <td>name=top</td>
        <td></td>
    </tr>
    <tr>
        <td>click</td>
        <td>//area[3]</td>
        <td></td>
    </tr>
```

```
<tr>
	<td>click</td>
	<td>//area[4]</td>
	<td></td>
</tr>
<tr>
	<td>selectWindow</td>
	<td>name=main</td>
	<td></td>
</tr>
<tr>
	<td>type</td>
	<td>name=pwd</td>
	<td>mrsoft</td>
</tr>
<tr>
	<td>selectWindow</td>
	<td>name=top</td>
	<td></td>
</tr>
<tr>
	<td>clickAndWait</td>
	<td>//area[5]</td>
	<td></td>
</tr>
</tbody></table>
</body>
</html>
```

(3) 回放生成的日志如图 11-22 所示。

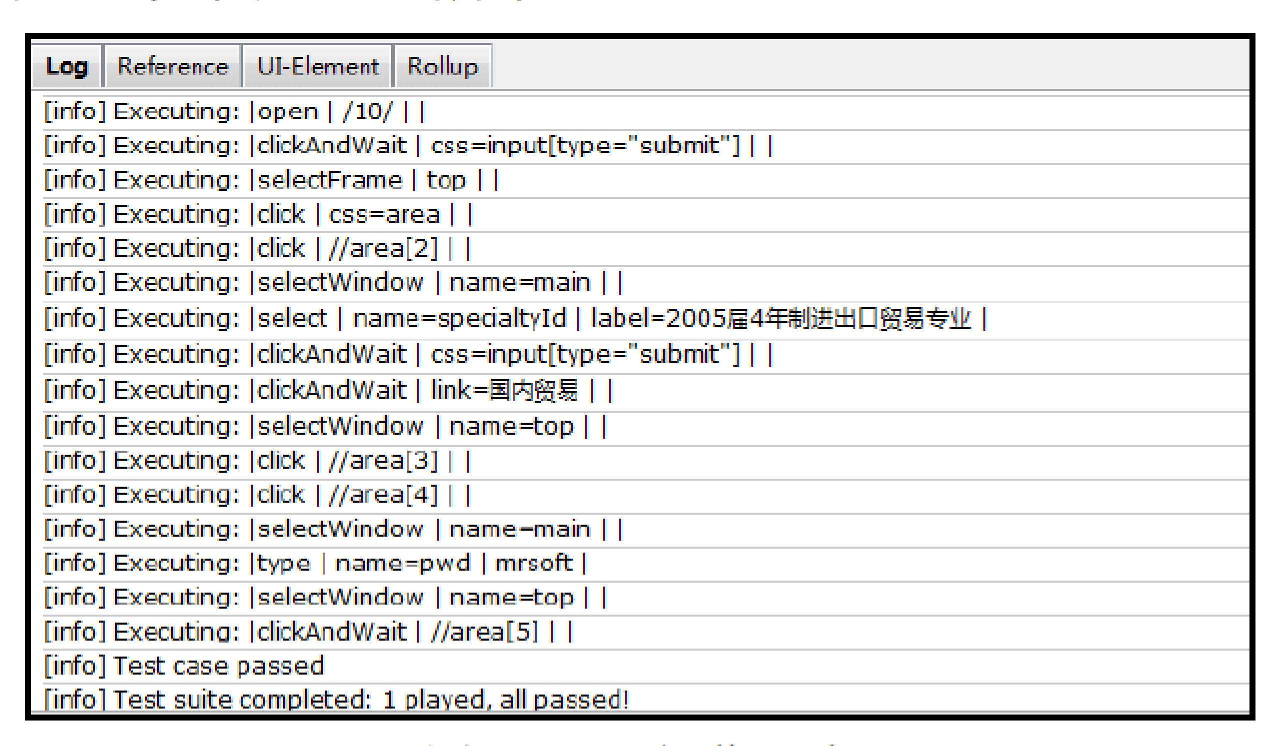


图 11-22　操作日志

(4) 回放的结果如图 11-23 所示。

4) 总结

Selenium 是软件工程师、设计人员和测试人员的工具箱中又一个有用且重要的工具。通过将该工具与持续集成工具相结合，团队就可以将验收测试自动化，并构建更好的软件，因为使用它们可以更容易、更早、更频繁地发现 Bug。Selenium 的另一个优点是可以节省时间，使开发人员和测试人员不必将时间花在本可以(也应该)自动化的手工任务上，从而让团队将精力放在更有价值的活动上。

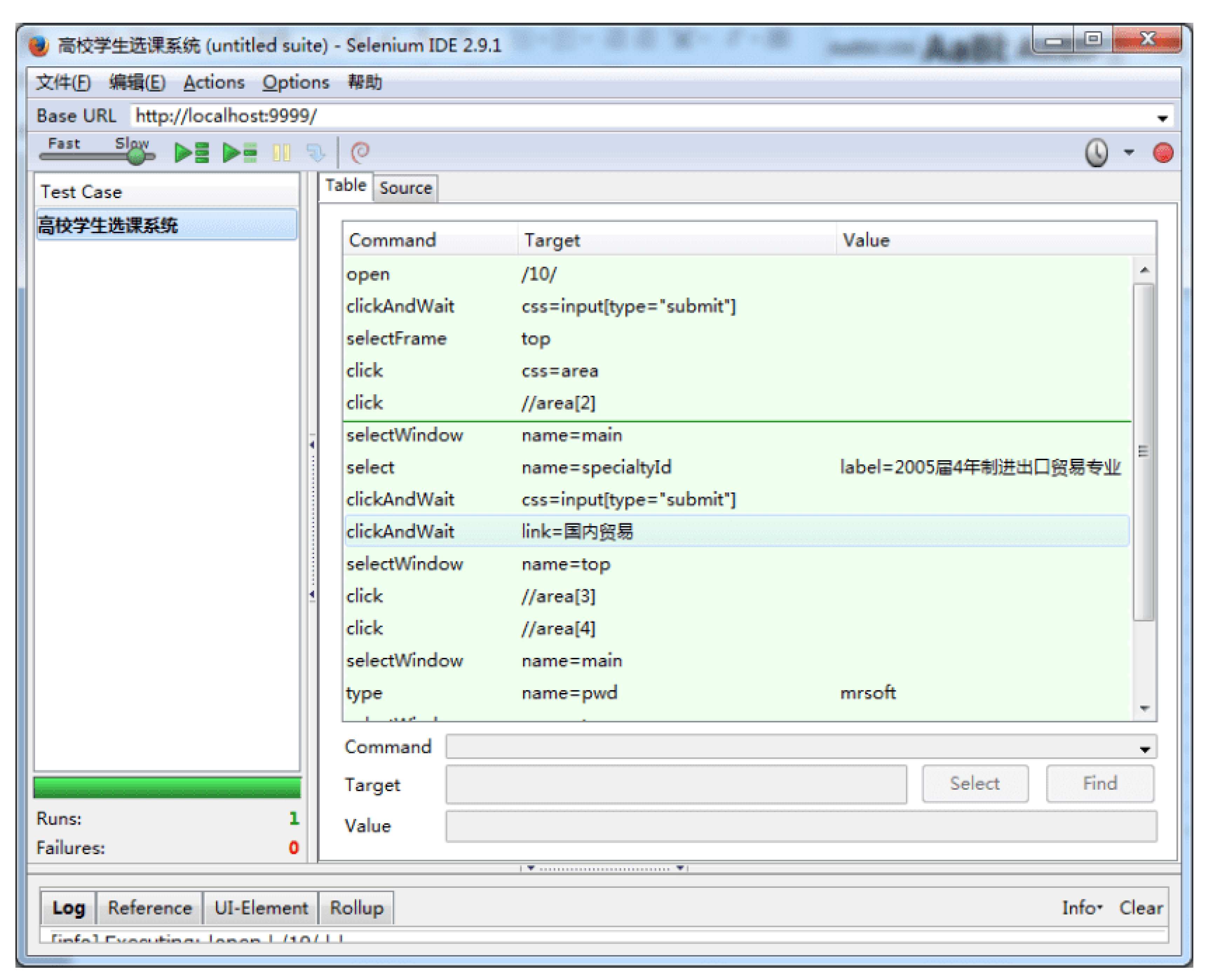


图 11-23　回放结果

11.2.2　性能自动化测试工具 JMeter 及其应用

1. JMeter 测试环境建立

搭建 JMeter 测试环境的过程非常简单，由于 JMeter 是一个图形界面配置工具，并且不像 LoadRunner 那样定位为高端测试人员专用，因此 JMeter 能够让普通 Web 应用开发人员快速上手。

1) 软件下载

搭建 JMeter 测试环境所需要下载的工具均是开源软件：

- JDK1.5：http://java.sun.com/javase/downloads/index.jsp
- Apache Tomcat 5.5：http://tomcat.apache.org/download-55.cgi
- jakarta-jmeter-2.2：http://jakarta.apache.org/site/downloads/downloads_jmeter.cgi
- sqljdbc_1.1.1501.101_enu.exe：http://www.microsoft.com/downloads/details.aspx?FamilyId=6D483869-816A-44CB-9787-A866235EFC7C&displaylang=en

2) 搭建 JMeter 测试环境

(1) 安装 JDK(1.4 以上)。

下载 jdk-1_5_0_09-nb-5_0-win-ml.exe，直接双击以默认方式安装，一般安装至 C:\Program Files\Java 目录下。

设置环境变量：右击“我的电脑”，执行“高级”|“环境变量”|“系统变量”。在“系统变量”框里进行环境变量设置。

① 新建变量 JAVA_HOME，值为：安装 JDK 的目录。

② 新建变量 CLASSPATH，值为：%JAVA_HOME%\lib; %JAVA_HOME%\bin。

③ 在 path 变量后加上：%JAVA_HOME%\lib;%JAVA_HOME%\bin。

在命令提示符窗口中输入 JAVA 或 JAVAC 后回车，如果出现帮助信息，则说明 JDK 安装成功。

(2) 安装 Apache Tomcat。

下载 apache-tomcat-5.5.20.exe(或更高版本)，直接双击按照默认路径安装，一般安装至 C:\Program Files\Apache Software Foundation\Tomcat 5.5 目录下。然后在 Tomcat 中部署一个简单的 Web 应用，这个 Web 应用是待测试的案例，在现实使用中这可能是一个已经存在的 Web 应用系统，也可能是一个正处于构建中的 Web 应用系统。11.1.2 节将详细说明这个 Web 应用案例。

(3) 安装 JMeter。

解压 jakarta-jmeter-2.2.zip 文件至 C:\jakarta-jmeter-2.2 目录下(当前的 JMeter 最新版本是 2.6)。

在桌面上右击“我的电脑”，执行“高级”|“环境变量”，然后执行“系统变量”|“新建”。在“变量名”中输入 JMETER_HOME，在“变量值”中输入 C:\jakarta-jmeter-2.2，在 CLASSPATH 变量值中添加%JMETER_HOME%\lib\ext\ApacheJMeter_core.jar、%JMETER_HOME%\lib\jorphan.jar 和%JMETER_HOME%\lib\logkit-1.2.jar，然后单击“确定”即可。

JMeter 有如下几个目录：bin、docs、extras、lib 和 printable_docs。进入 bin 目录，运行下面的 jmeter.bat 就可以看见 JMeter 的 GUI 客户端了，可以对测试进行相关的配置。如果要对自己开发的网站进行测试，则需要运行网站应用服务器，然后将访问地址录入到 JMeter 中进行测试。

lib 目录下有两个子目录，一个是 ext 目录，另一个是 junit 目录。ext 目录用于存放用户对 JMeter 进行扩展的测试应用，而用户的扩展所依赖的包则要直接放在 lib 目录下(而不是 lib/ext 目录下)。JMeter 会自动从它的/lib 和/lib/ext 目录的 jar 包中发现类。如果用户开发了新的 JMeter 组件，可将它们以 jar 包形式复制到 JMeter 的/lib/ext 目录下。JMeter 将会自动发现这里的任何 jar 包的 JMeter 组件。

(4) JMeter 针对不同的 Web 应用测试可能还需要 SSL 加密、JDBC 驱动、Apache SOAP 以及 BeanShell 等包。

① SSL 加密：为了测试一台使用 SSL 加密(HPPS)的 Web 服务器，JMeter 需要提供一个 SSL 实现(例如 Sun 的 Java Secure Sockets Extension(JSSE))，包含需要的加密包到 JMeter 的 CLASSPATH 中。同时，通过注册 SSL 提供者更新 system.properties 文件。

② JDBC 驱动：如果需要做 JDBC 测试，则需要添加厂商的 JDBC 驱动到 CLASSPATH 中。确认文件是 jar 文件而不是 zip 文件。

③ Apache SOAP：Apache SOAP 需要 mail.jar 和 activation.jar。这时需要下载并复制这两个 jar 文件到 jmeter/lib 目录下。一旦文件放到那里，JMeter 就会自动找到它们。

④ BeanShell：为了运行 BeanShell 函数或任何 BeanShell 测试元件(取样器、定时器等)，需要从 http://www.beanshell.org/下载 BeanShell 的 jar 文件并将其复制到 jmeter/lib 目录下，JMeter 会自动找到它。

2. JMeter 测试功能及使用流程

1) JMeter 的主要功能

JMeter 的主要功能包括：

(1) 能够对 HTTP 和 FTP 服务器进行压力测试和性能测试，也可以对任何数据库进行同样的测试(通过 JDBC)。

(2) 完全的可移植性和 100%纯 Java。

(3) 完全 Swing 和轻量组件支持包(预编译的 JAR 使用 javax.swing.*)。

(4) 完全多线程。框架允许通过多个线程并发取样，以及通过单独的线程组对不同的功能同时取样。

(5) 精心的 GUI 设计允许快速操作和更精确的计时。

(6) 缓存和离线分析/回放测试结果。

JMeter 同时还具有高可扩展性：

(1) 可链接的取样器允许无限制的测试能力。

(2) 各种负载统计表和可链接的计时器可供选择。

(3) 数据分析和可视化插件提供了很好的可扩展性及个性化。

(4) 具有提供动态输入到测试的功能(包括 JavaScript)。

(5) 支持脚本编程的取样器(在 1.9.2 及以上版本中支持 BeanShell)。

2) JMeter 的主要元件

JMeter 的主要元件有：

(1) 测试计划是使用 JMeter 进行测试的起点，是其他 JMeter 测试元件的容器。

(2) 线程组代表一定数量的并发用户，可以用来模拟并发用户发送请求。实际的请求内容在 Sampler 中定义，它被线程组包含。

(3) 监听器负责收集测试结果，同时也被告知结果显示的方式。

(4) 逻辑控制器可以自定义 JMeter 发送请求的行为逻辑，它与 Sampler 结合使用可以模拟复杂的请求序列。

(5) 断言可以用来判断请求响应的结果是否正如用户所期望的。它可以被用来隔离问题域，即在确保功能正确的前提下执行压力测试。这个限制对于有效的测试是非常有用的。

(6) 配置元件负责维护 Sampler 所需要的配置信息，并根据实际的需要修改请求的内容。

(7) 前置处理器和后置处理器负责在生成请求之前和之后完成工作。前置处理器常常用来修改请求的设置，后置处理器则常常用来处理响应的数据。

(8) 定时器负责定义请求之间的延迟间隔。

3) JMeter 的测试流程

在测试环境搭建好之后，就可以开始进行测试了。使用 JMeter 进行测试主要包含以下几个步骤，当然最初 JMeter 必须是运行着的，如图 11-24 所示(在 JMeter 的 bin 目录下找到 jmeter.bat 批处理文件，双击打开)。

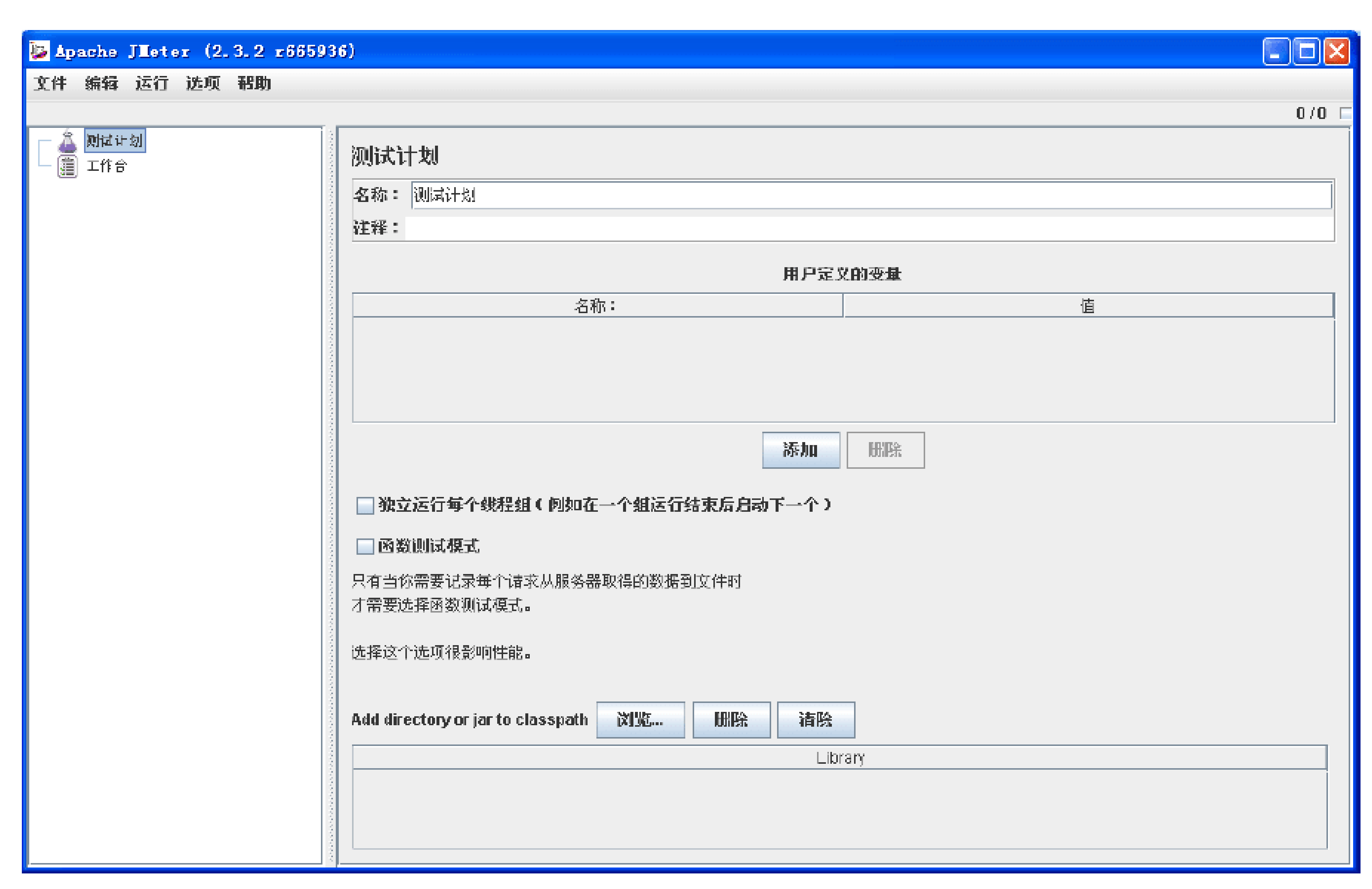


图 11-24　启动 JMeter

(1) 建立测试计划

测试计划描述了测试执行过程中 JMeter 的执行过程和步骤，一个完整的测试计划包括一个或多个线程组(Thread Groups)、逻辑控制器(Logic Controller)、样例产生控制器(Sample Generating Controllers)、监听器(Listener)、定时器(Timer)、断言(Assertions)、配置元素(Config Elements)。启动 JMeter 时，它已经建立了一个默认的测试计划。

一个 JMeter 应用实例只能建立或打开一个测试计划。

(2) 添加线程组

线程组是测试计划中最先建立的部分。在线程组中，测试者将要指定同时将有多少个线程并发访问；每两个线程访问之间的间隔时间(Ramp-Up Period)；总共要循环访问多少次，可以选择具体数字，也可以选择无限期循环，如图 11-25 所示。

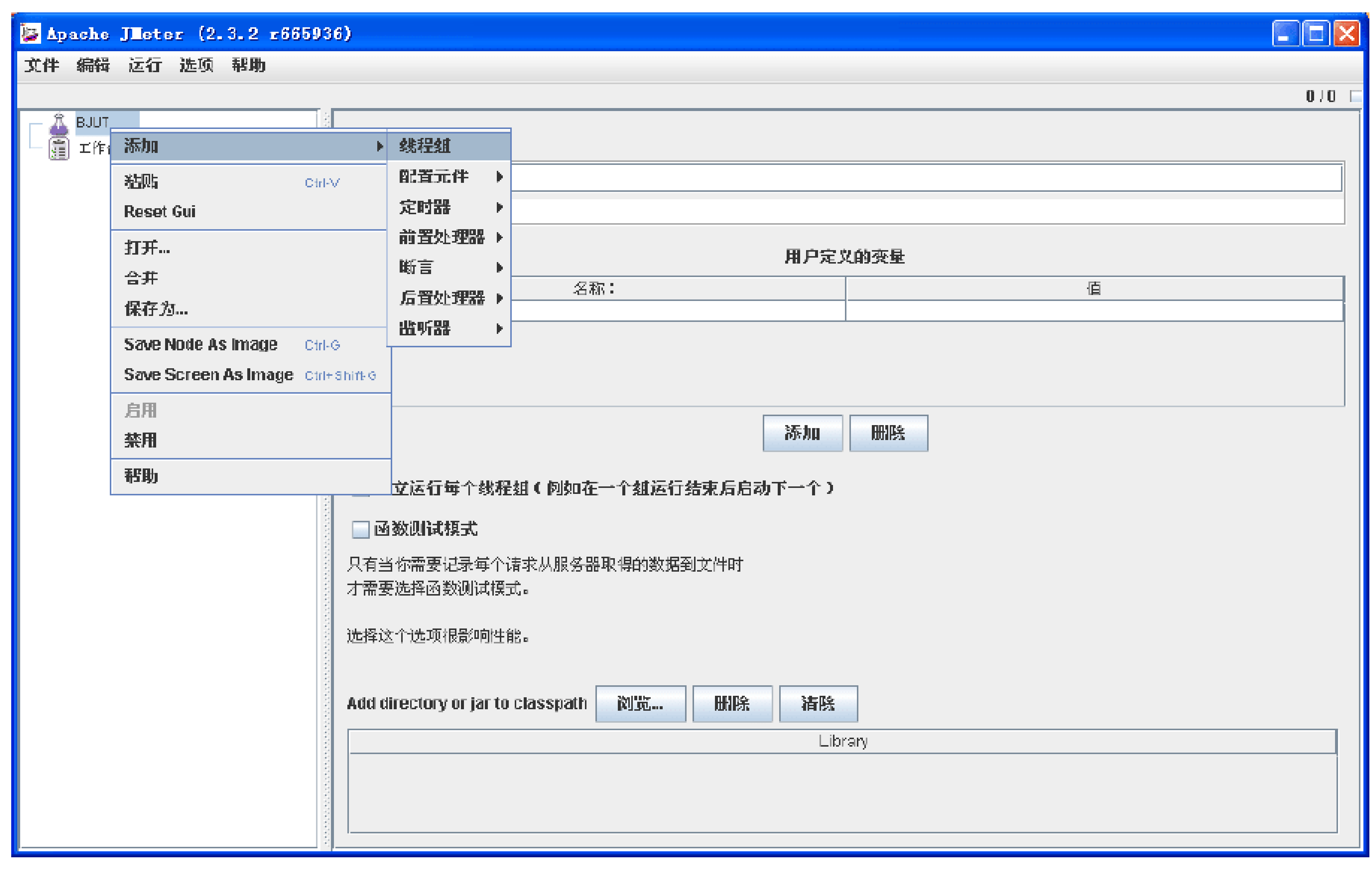


图 11-25　线程组的选择及配置



图 11-25(续)

(3) 添加取样器

请求是 Web 测试的主体，通过向指定服务器提交特定请求并捕捉返回状态，来检测 Web 应用的性能。JMeter 中可添加的特定请求由取样器来选择和发送，其中包括 HTTP 请求、FTP 请求、JDBC 数据库访问请求、LDAP 请求、AJP 1.3 请求、SOAP 请求、JMS 请求等，如图 11-26 所示。本章主要展示 HTTP 请求的测试过程。

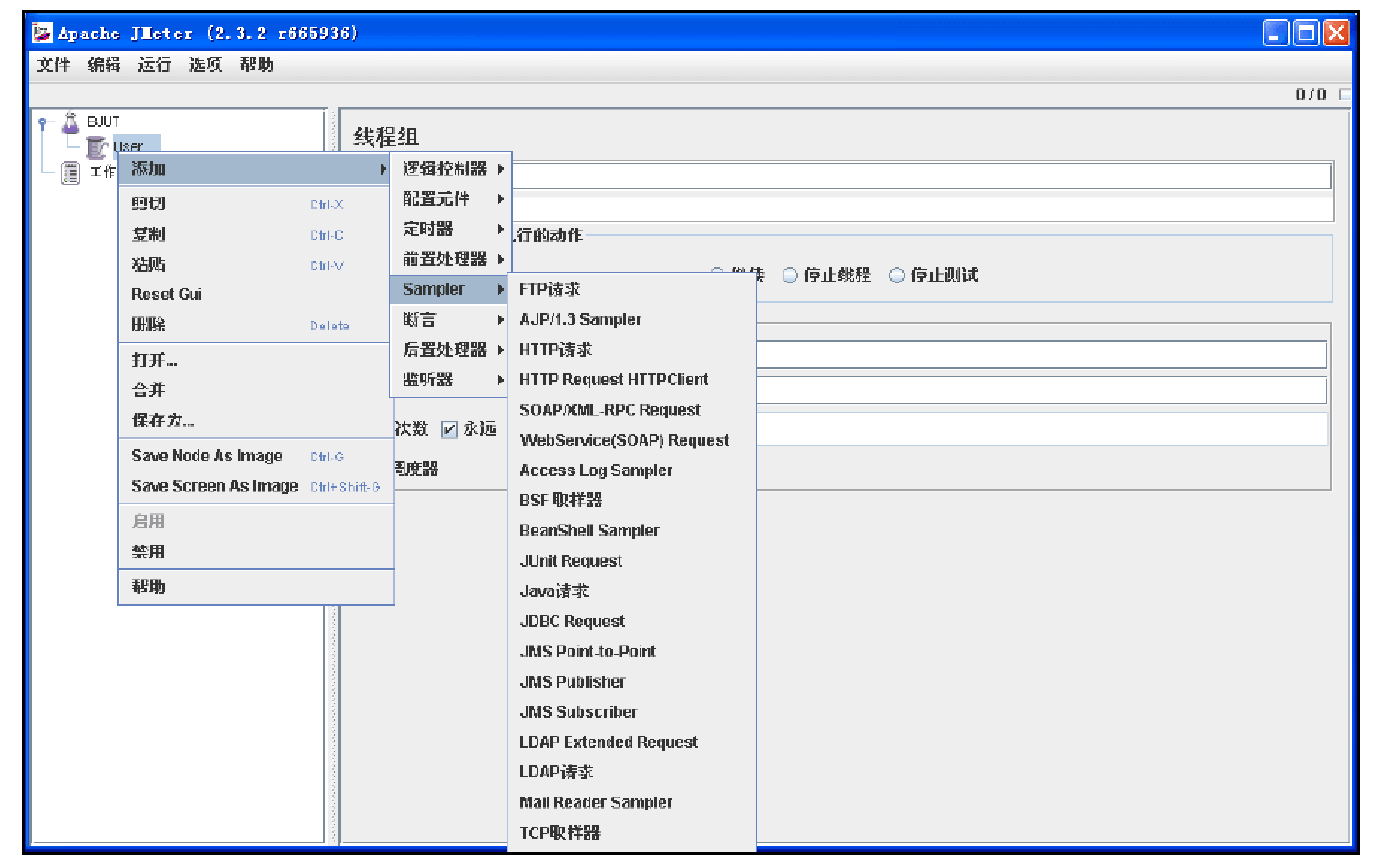


图 11-26　取样器的可选内容

每个取样器都有一些可以设置的属性。可以通过添加一个或多个配置元件到取样器来进一步定制它。注意 JMeter 发送请求是按照取样器出现在树中的顺序来进行的。

如果想发送多个类型相同的请求(例如 HTTP 请求)到相同的服务器，可以考虑使用一个默认配置元件。每个控制器都有一个或多个默认配置元件。

记着添加一个监听器到线程组中来查看或保存请求结果至磁盘。

(4) 添加监听器

为了记录测试信息及使用 JMeter 提供的可视化界面查看测试结果，监听器提供了许多种结果分析方式以供选择，测试者可以根据自己习惯的分析方式来选择不同的结果显示方式，如图 11-27 所示。以上是进行 JMeter 性能测试的必要步骤，JMeter 同时还存在一些可选步骤。

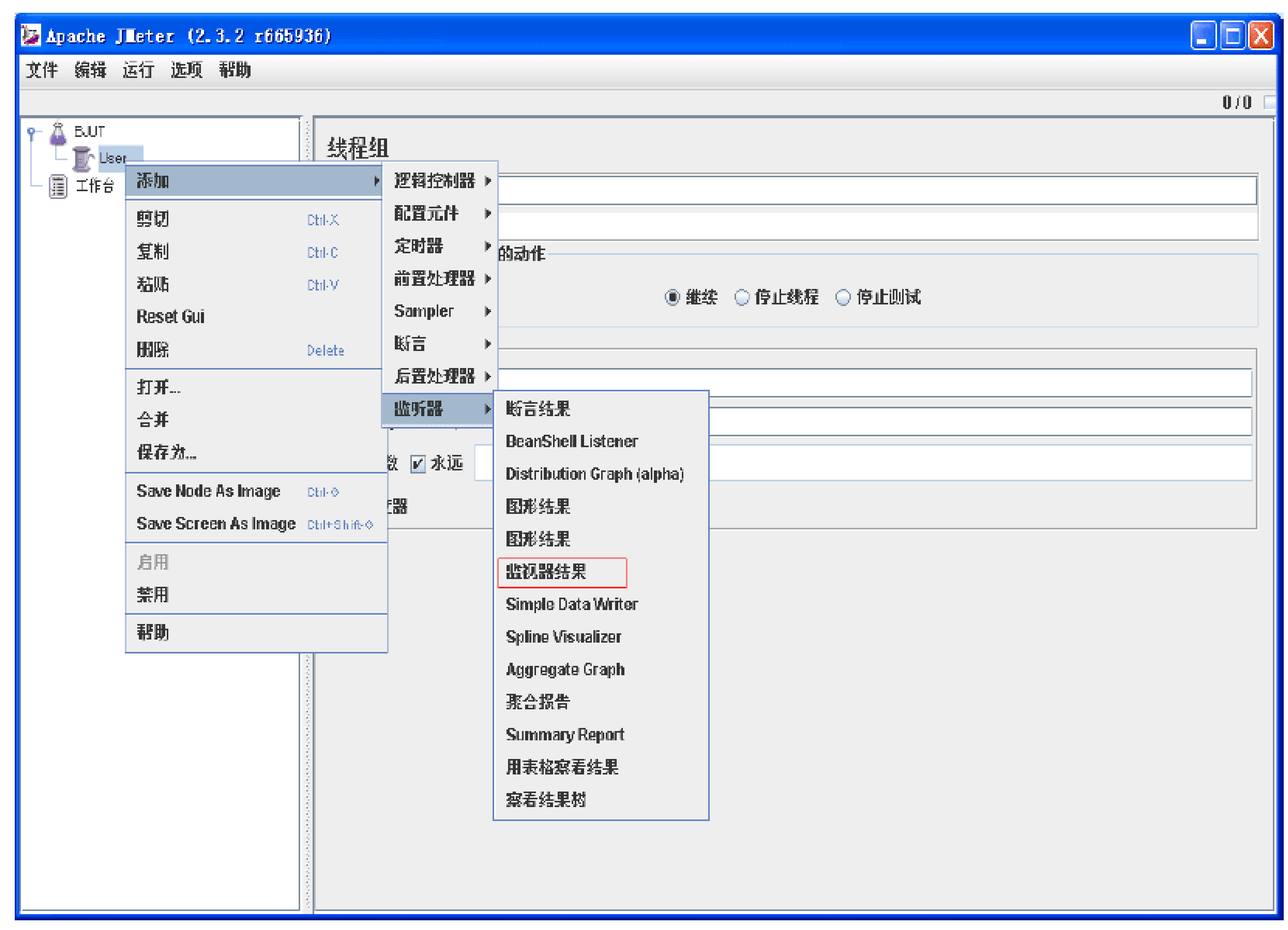


图 11-27　监听器的可选内容

添加逻辑控制器：逻辑控制器允许测试人员定制当发送请求时 JMeter 使用的判断逻辑，如图 11-28 所示。逻辑控制器还可以作为下列任何元件的子元件：取样器(请求)、配置元件及其他逻辑控制器。逻辑控制器可以改变来自它们的子元件的请求顺序。它们还可以修改请求本身，从而导致 JMeter 重复请求。

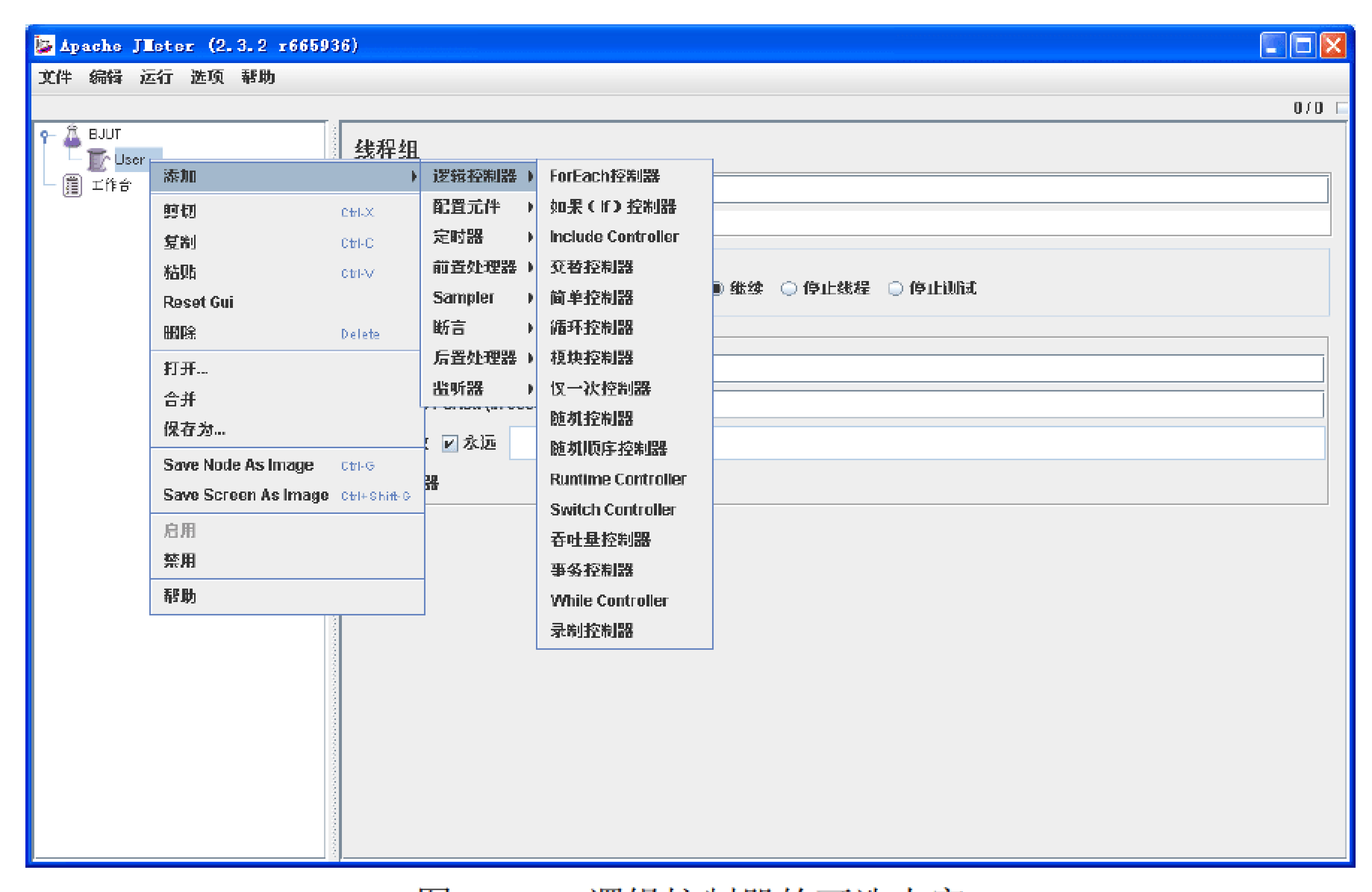


图 11-28　逻辑控制器的可选内容

配置元组：实际的测试工作往往是针对同一台服务器上的 Web 应用展开的，所以 JMeter 提供了这样一种设置，就是将默认请求属性设置成被测服务器的相关属性，如图 11-29 所示。在以后的同等请求设置中就可以忽略这些相同参数的设置，以减少设置参数录入的时间。

添加断言：断言允许测试人员给出关于从测试服务器接收到的响应的行为，如图 11-30 所示。使用断言可以测试应用程序是否返回期望的结果。

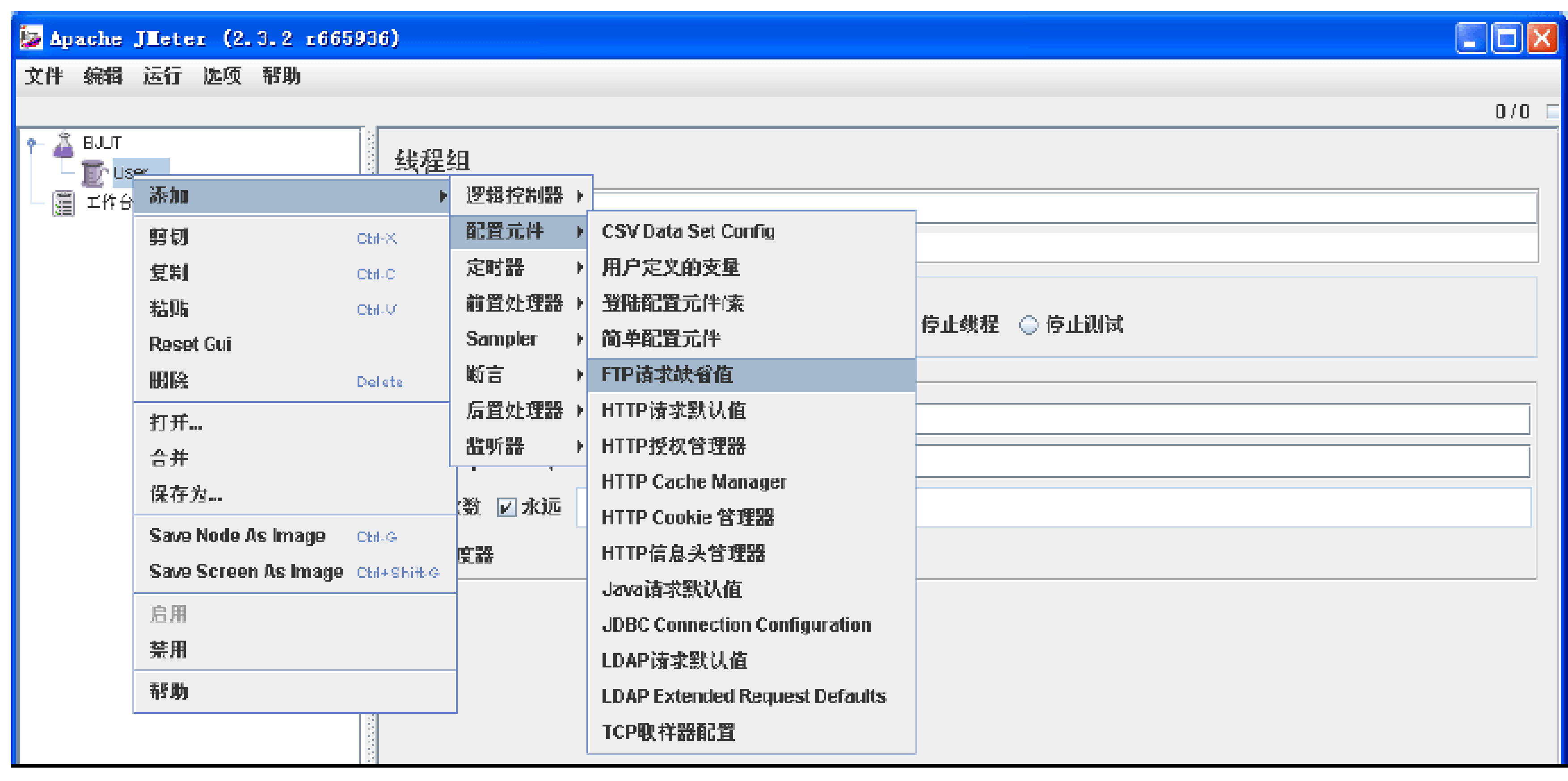


图 11-29　可配置的默认请求

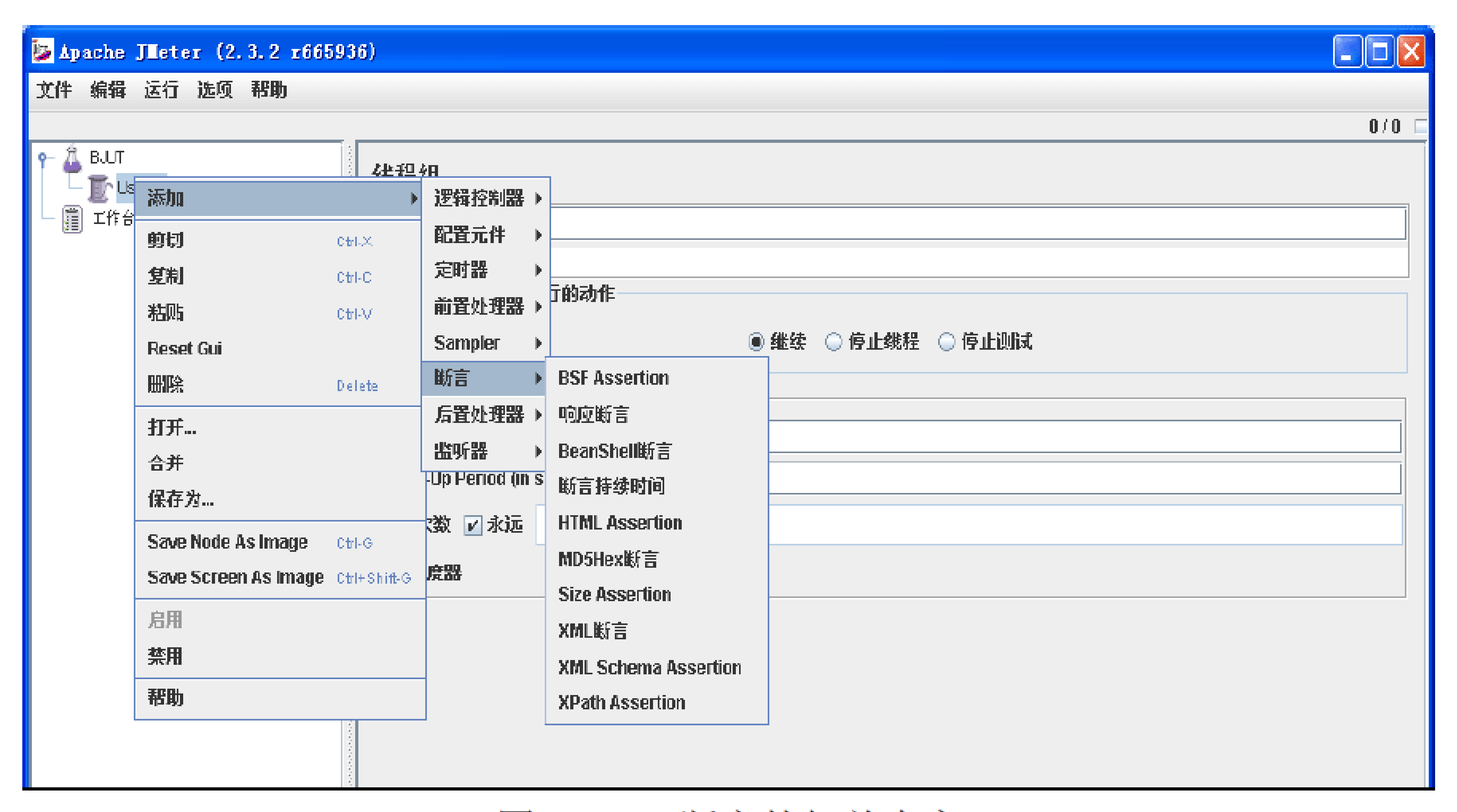


图 11-30　断言的相关内容

添加定时器：定时器可以让测试在指定的时间进行，也可以在指定的时间周期之后才开始运行，如图 11-31 所示。默认情况下，JMeter 线程发送请求时不请求暂停。建议通过添加一个可用的定时器到线程组中来指定延迟。如果不添加延迟，JMeter 会在短时间内产生太多请求，这可能会压垮测试中的服务。

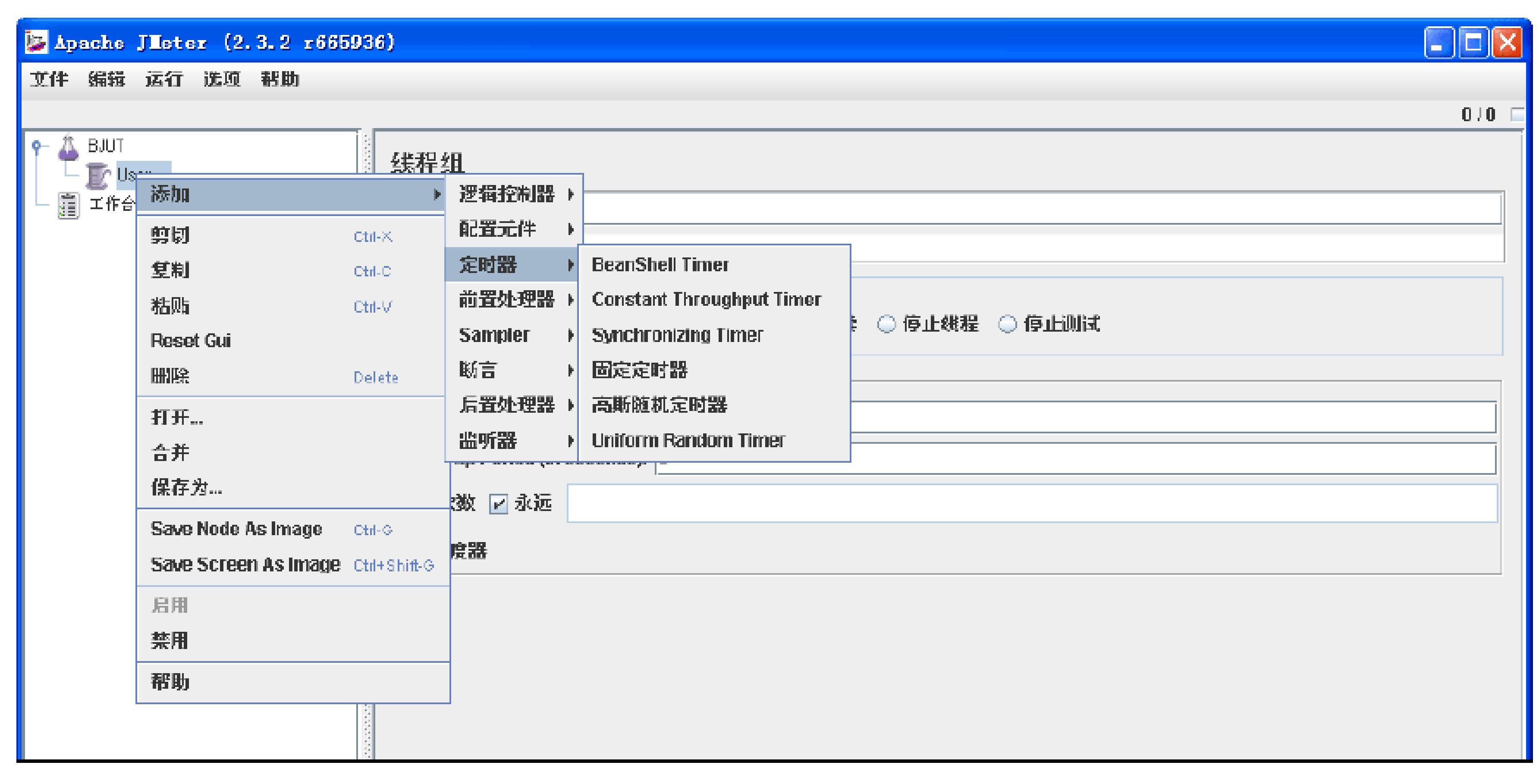


图 11-31　定时器的可选内容

定时器会使 JMeter 在一个线程开始每个请求时延迟一段时间。如果选择添加多于一个定时器到线程组中，JMeter 会在执行取样器前获得定时器数量并暂停那个时间量。

添加前置处理器和后置处理器：前置处理器和后置处理器可以在请求的开始之前和结束之后分别对请求做一些特定的操作，比如 URL 重写，如图 11-32 所示。

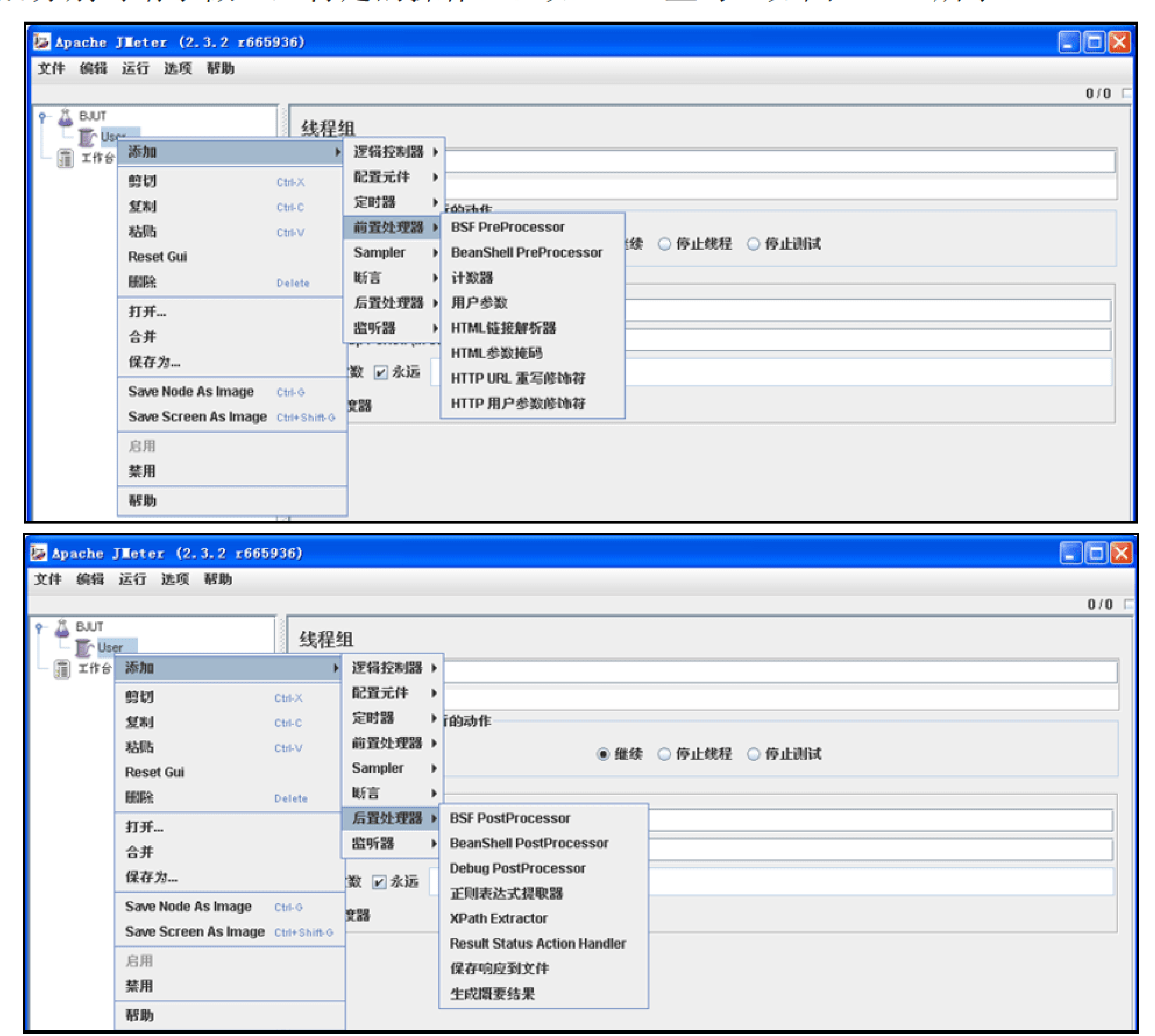


图 11-32 前置处理器和后置处理器的可选内容

JMeter 的执行顺序是：定时器、取样器、后置处理器(如果 SampleResult 不为空)、断言(如果 SampleResult 不为空)、监听器(如果 SampleResult 不为空)。

3. JMeter 测试应用举例

下面针对 JMeter 的几种典型测试，分别给出相应的例子。

1) 测试 HTTP 请求

创建 4 个用户，分别向高校学生选课管理系统网站上的两个网页发送请求。每个用户运行测试两次。这样，总的 HTTP 发送请求次数为 16(4 个用户×2 次请求×重复 2 次)。

(1) 添加用户

处理 JMeter 测试计划的第一步就是添加线程组元件。这个线程组会模拟用户数量，用户应该发送请求的频率以及应该发送的数量。

下面来添加一个线程组：首先选择这个测试计划，用鼠标右击它，然后在弹出的菜单中选择“添加”|“线程组”命令；首先为这个线程组起一个有意义的名字，在 Name 文本框中输入“高校学生选课系统”；下一步，增加用户的数量(线程)为 4；在 Ramp-Up Period

文本框中，使用默认值 0，该属性表示每个用户启动的迟延时间；最后，禁用 Forever 复选框并设置循环次数为 2，该属性表示测试的重复次数，如图 11-33 所示。

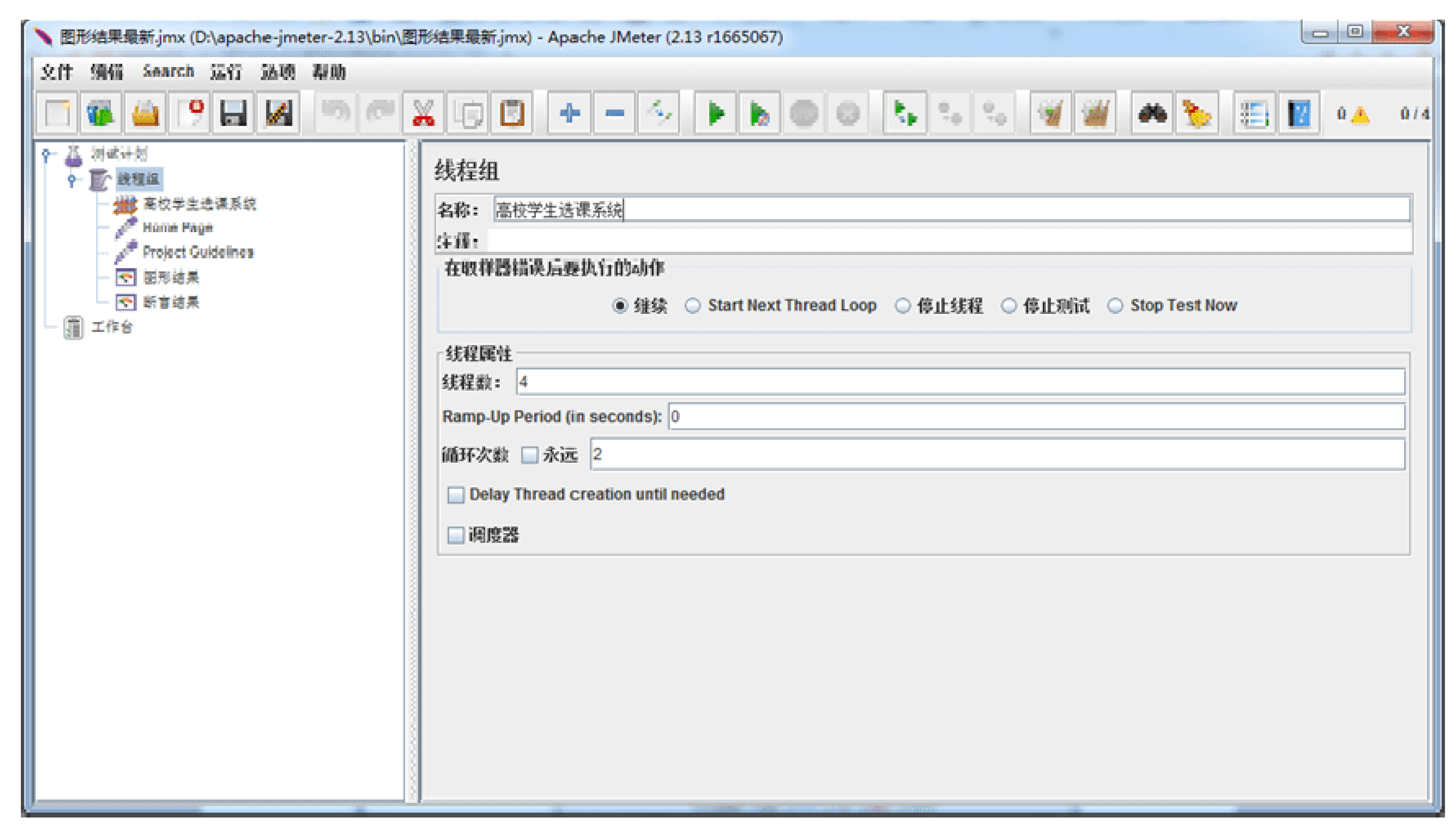


图 11-33　线程组参数设置

(2) 添加 HTTP 请求默认属性

首先选择“高校学生选课系统”元件，用鼠标右击并在弹出的菜单中选择“添加”|“配置元件”|“HTTP 请求默认值”命令(HTTP 请求默认值元件并不告诉 JMeter 发送 HTTP 请求，它仅定义这个 HTTP 请求所使用的默认值)。然后选择这个新元件，以显示其控制面板。

对于创建的测试计划，所有的 HTTP 请求都将被发送到相同的 Web 服务器地址。

(3) 添加 Cookie 支持

除非应用程序明确不使用 Cookie，否则几乎所有的网站应用程序都会使用 Cookie 支持。要添加 Cookie 支持，可以简单地在测试计划中为每一个线程组添加一个 HTTP Cookie 管理器，以确保每个线程组都有自己的 Cookie，并能共享跨越所有的 HTTP 请求对象。

要添加 HTTP Cookie 管理器，可选择这个线程组，选择“添加”|“配置元件”|“HTTP Cookie 管理器”，也可以从“编辑”菜单或通过鼠标右击来实现添加。

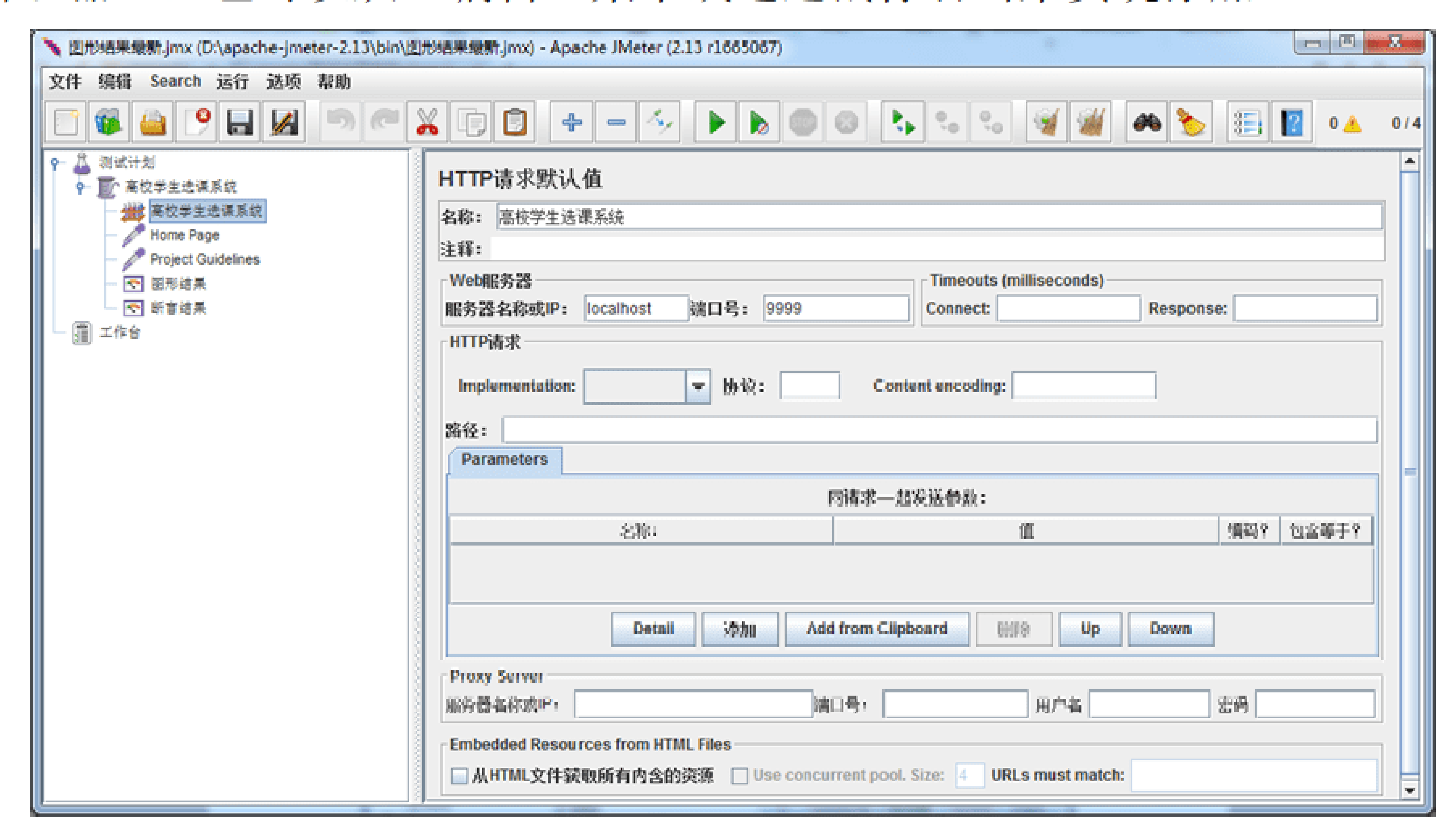


图 11-34　设置 HTTP 请求属性

(4) 添加 HTTP 请求

在这个测试计划中需要实现两个 HTTP 请求：一个是高校学生选课系统网站首页，另一个是工程向导网页。JMeter 将按照它们在树中出现的次序来发送请求。

首先来为高校学生选课系统元件添加第一个 HTTP 请求(“添加” | “取样器” | “HTTP 请求”)。然后从树中选择 HTTP 请求元件并修改相关属性：更改名称为 Home Page；设置路径为/。不必设置服务器名，因为已在 HTTP 默认请求元件中指定了这个值。然后添加第二个 HTTP 请求并修改相关属性：更改名称为 Project Guidelines；设置路径为/10，如图 11-35 所示。

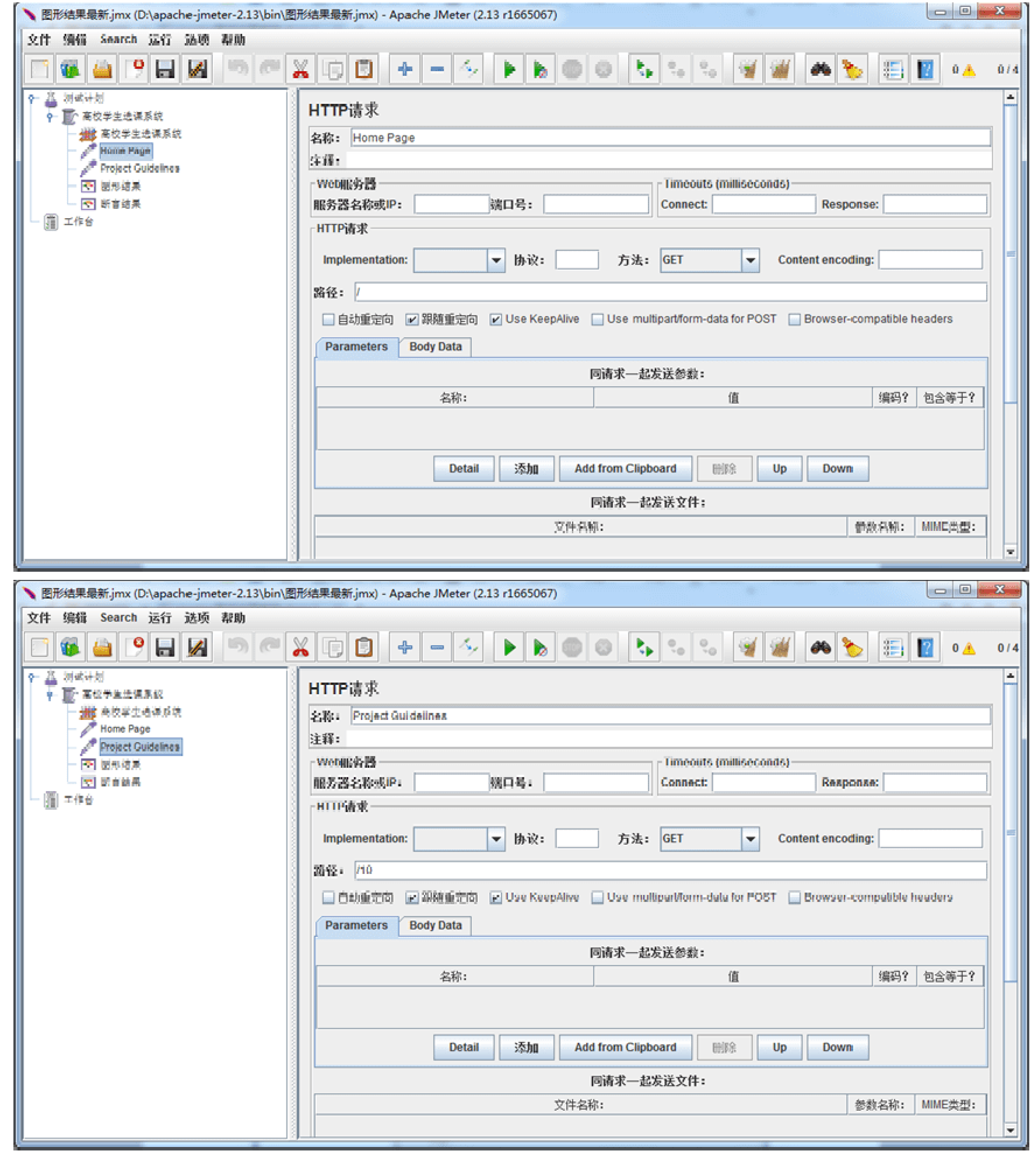


图 11-35　设置 HTTP 请求

5) 添加一个监听器来浏览/存储测试结果

监听器用于将所有的 HTTP 请求结果存储在一个文件中并显现出数据的可视模型。选择高校学生选课系统性能测试元件，然后添加一个图形结果监听器(“添加” | “监听器” | “图形结果”)。接着，指定文件路径和输出文件名，如图 11-36 所示。

标题解释：样本数目(No of Samples)是总共发送到服务器的请求数；最新样本(Latest Sample)是代表时间的数字，是服务器响应最后一个请求的时间；吞吐量(Throughput)是服务器每分钟处理的请求数；平均(Average)是总运行时间除以发送到服务器的请求数；中值

(Median)是代表时间的数字，有一半的服务器响应时间低于该值，而另一半高于该值；Deviation(偏离)表示服务器响应时间变化、离散程度测量值的大小，或者换句话说，就是数据的分布。

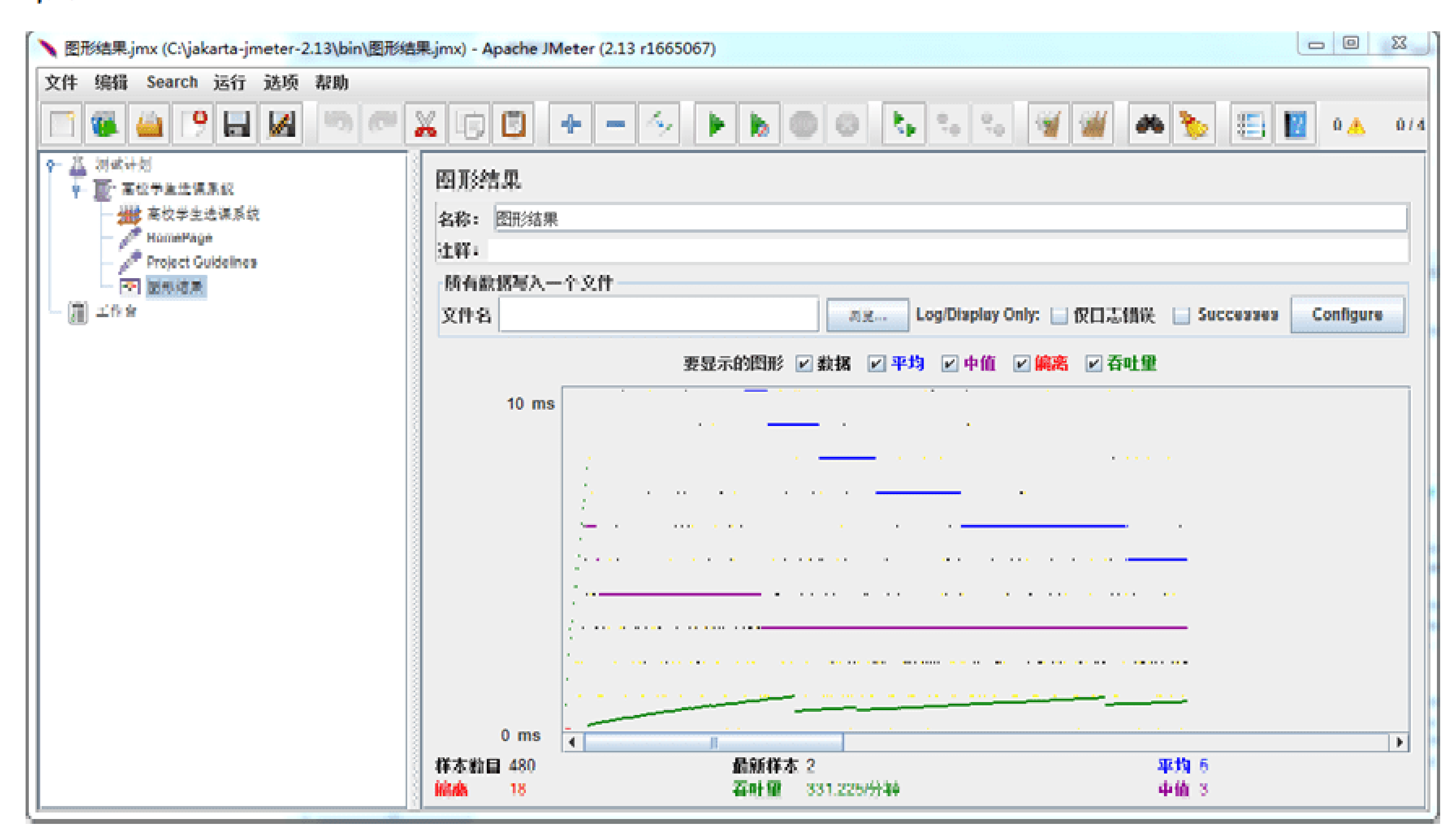


图 11-36　HTTP 请求监听的图形结果

2) FTP 测试

为 Apache 的 FTP 站点上的两个文件创建 4 个发送请求的用户，每个用户运行两次，所以整个测试有(4 个用户)*(2 个请求)*(重复 2 次)=16 个 FTP 请求，如图 11-37(a)所示。

这里选择进行测试的 FTP 是 Apache 的镜像站点 ftp://ftp.ring.gr.jp，如图 11-37 (b)所示。

(1) 添加用户和添加默认 FTP 请求配置基本上与 HTTP 测试的操作相同。

(2) 添加 FTP 请求。

在测试计划中需要制作两个 FTP 请求。一个是 CHANGES 文件(ftp://ftp.ring.gr.jp/pub/net/apache/httpd/CHANGES_2.0)，另一个是 HEADER 文件(ftp://ftp.ring.gr.jp/pub/net/apache/httpd/HEADER.html)。同样，JMeter 按照它们在树中出现的顺序来发送请求。

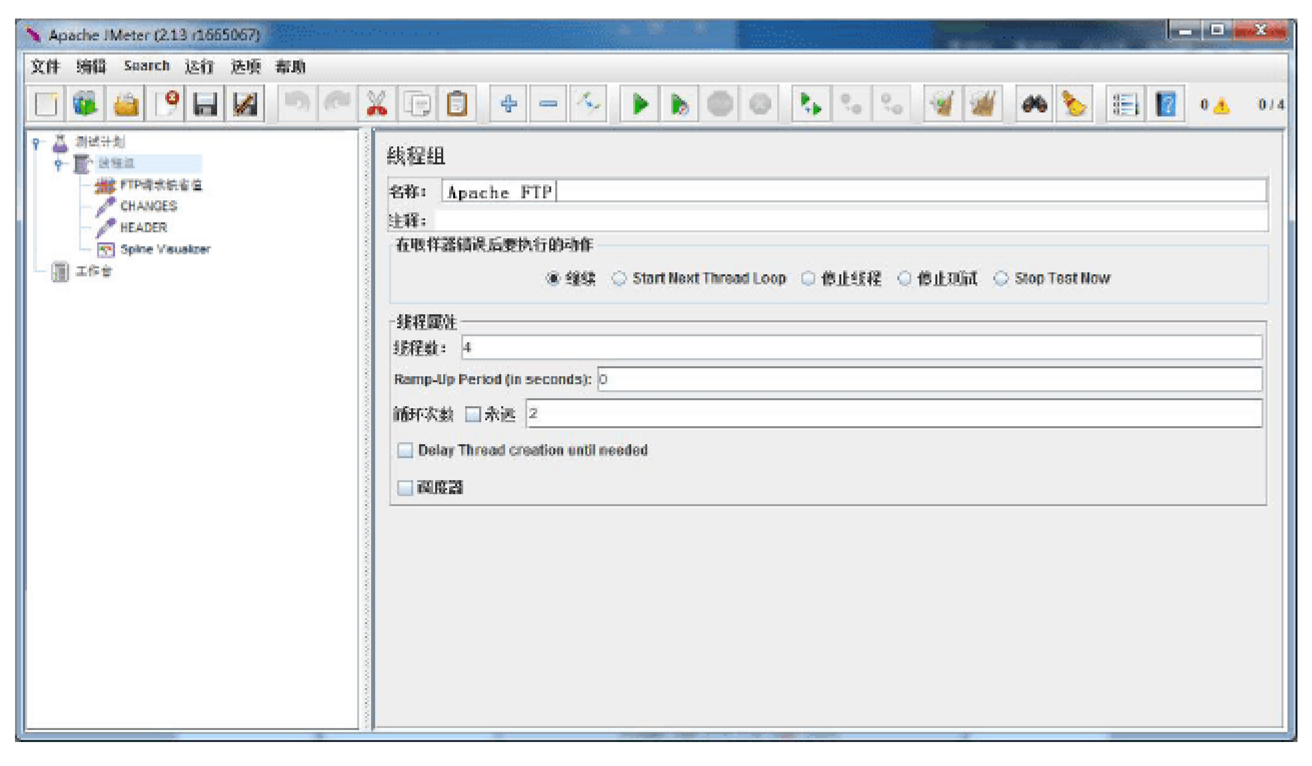
首先添加第一个 FTP 请求到 Apache FTP 元件(“添加” | “取样器” | “FTP 请求”)，然后在树中选择 FTP 请求元件，并编辑以下属性：修改名称为 CHANGES；修改 Remote File 文本框为 pub/net/apache/httpd/CHANGES_2.0，修改用户名为 anonymous；修改密码为 anonymous，如图 11-37(c)所示。

然后添加第二个 FTP 请求，并修改以下几个属性：修改名称为 Header.html，修改 Remote File 文本框为 pub/net/apache/httpd/HEADER.html，修改用户名为"anonymous"，修改密码为"anonymous"，如图 11-37(d)所示。

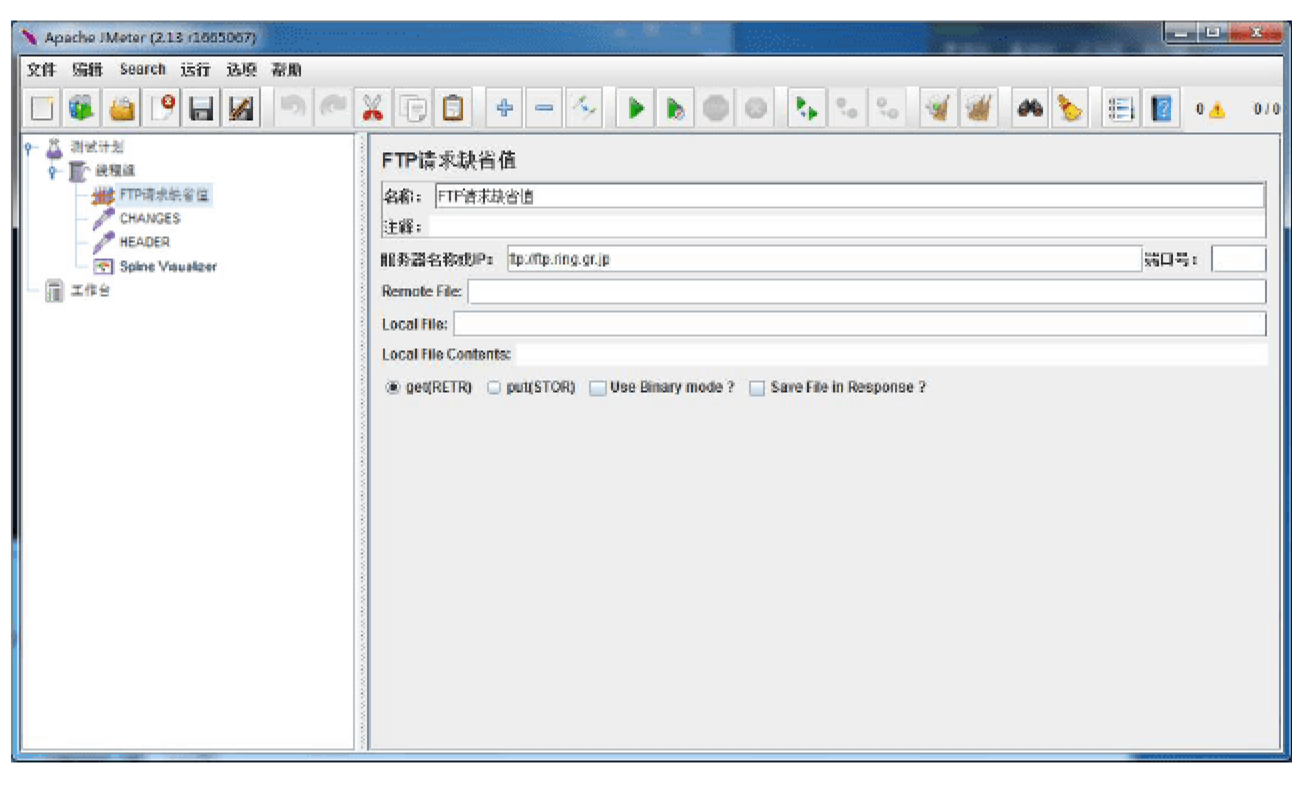
(3) 添加监听器来浏览/存储测试结果。

添加到测试计划的最后元件是监听器。这个元件负责存储所有的 FTP 请求结果到文件中，并展示一个可视化的数据模型。

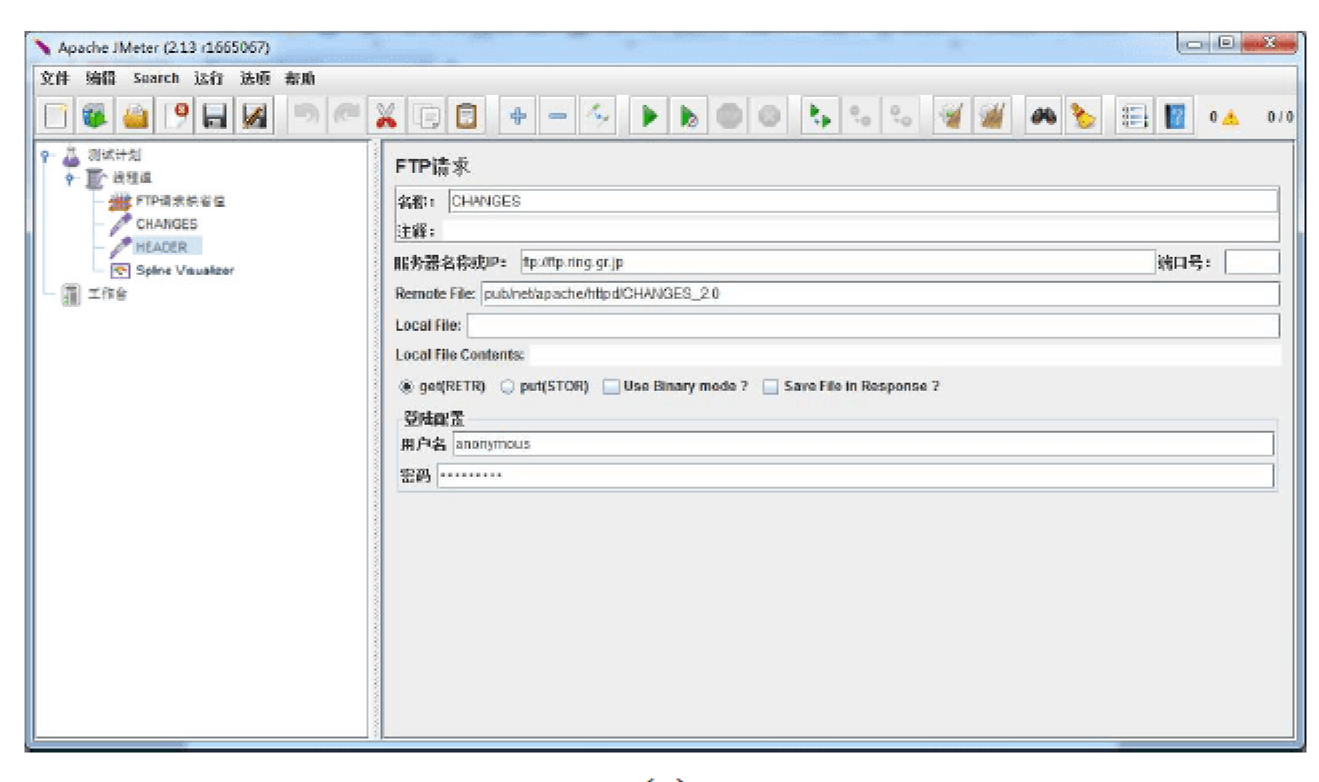
选择 Apache FTP 元件，添加一个 Spline Visualizer 监听器，如图 11-38 所示。



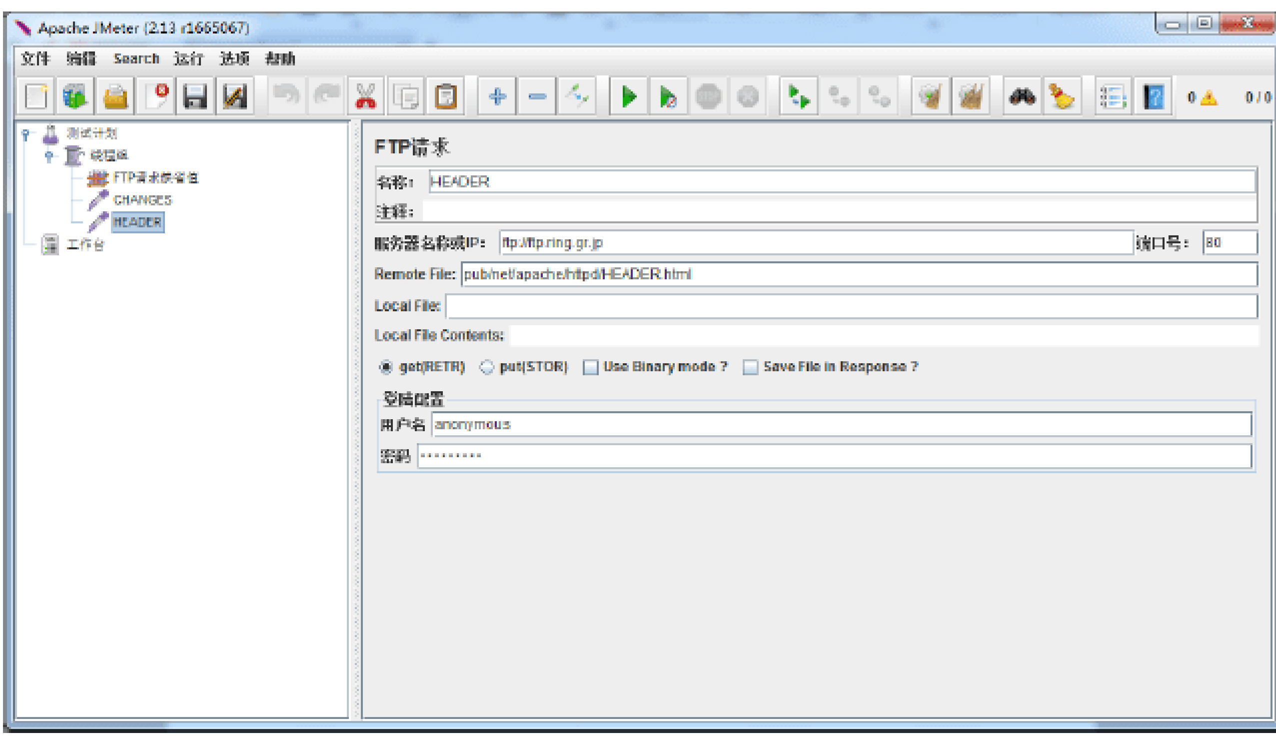
(a)



(b)



(c)



(d)

图 11-37　FTP 请求设置

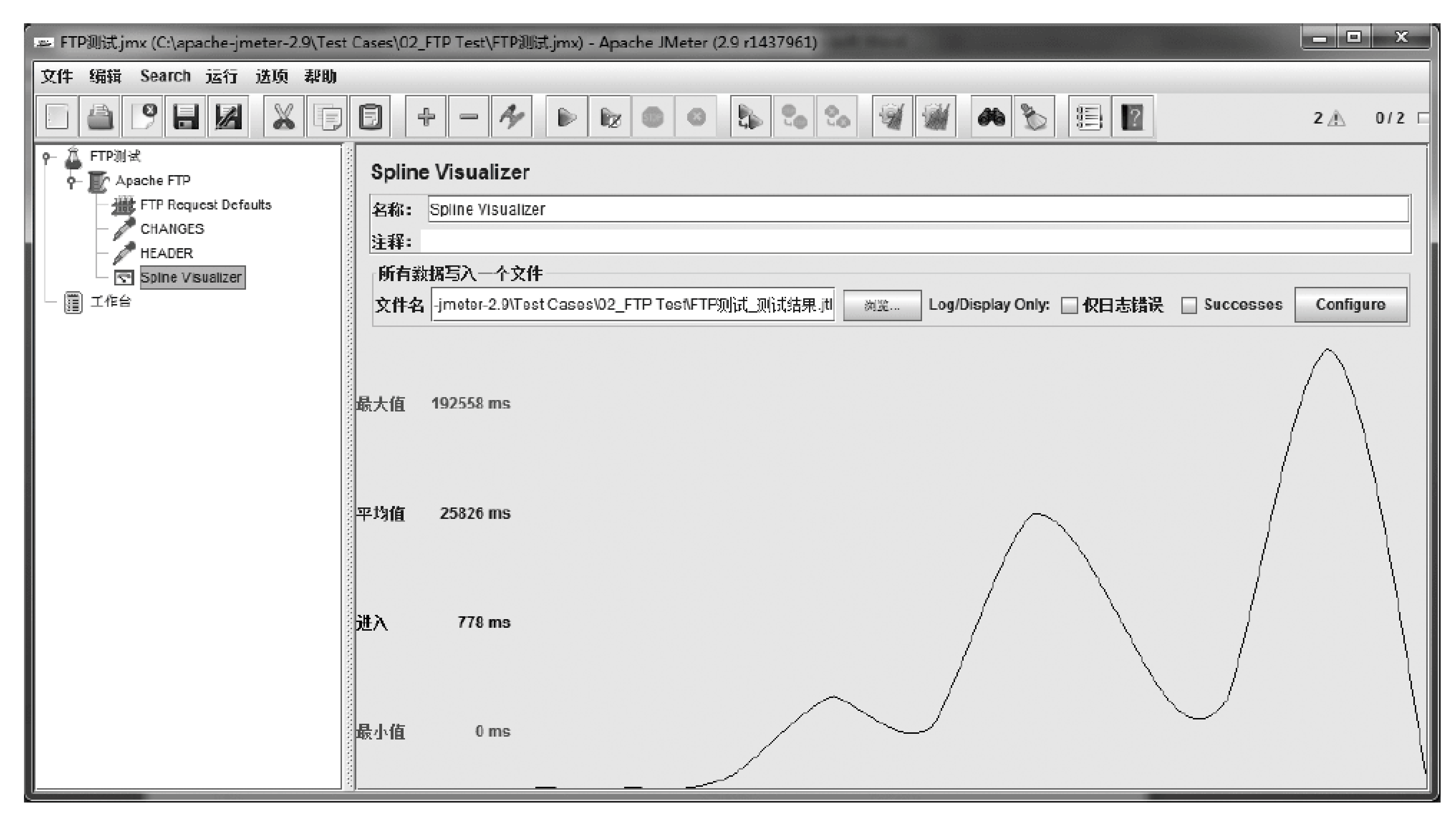


图 11-38　FTP 请求监听曲线结果

3) 数据库测试

创建 10 个用户来向数据库服务器发送两次 SQL 请求，总的 JDBC 请求数量就是(10 用户)×(两次请求)×(重复三次)=60。

这里使用了 MySQL 数据库驱动。要使用这个驱动，它所包含的.jar 文件就必须复制到%JMETER_HOME%\lib 目录下。

(1) 添加用户和添加默认 JDBC 请求配置基本上与 HTTP 测试的操作相同。

(2) 添加 JDBC 请求。

首先选择 JDBC 用户元件，用鼠标右击它，在弹出的菜单中选择“添加”|“配置元件”|“JDBC 连接配置”命令。接着，选择这个新元件以显示它的控制面板，如图 11-39 所示。

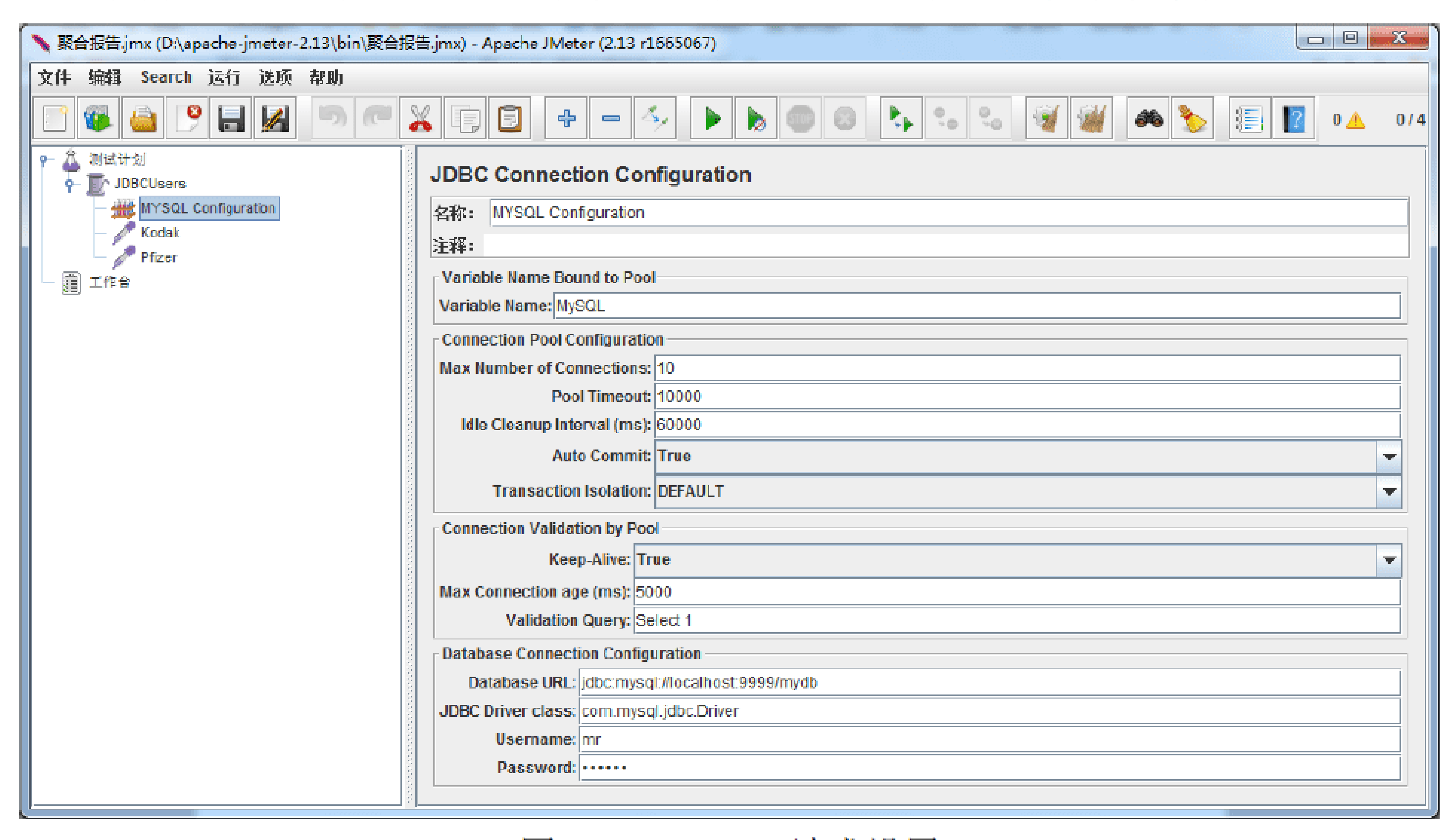


图 11-39　JDBC 请求设置

设置以下文本框(这里假定使用一个名为 test 的本地 MySQL 数据库)：

① Variable Name Bound to Pool(绑定到池变量)：需要能够唯一标识这个配置，用于

JDBC 取样器的识别。

② Database URL(数据库 URL)：jdbc: mysql://localhost:3306/mydb。

③ JDBC Driver class(JDBC 驱动类)：com.mysql.jdbc.Driver。

④ Username(用户名)：mr。

⑤ Password(密码)：111111。

再次选择 JDBC 用户元件，用鼠标右击它，在弹出的菜单中选择“添加”|“Sampler”|“JDBC 请求”命令。

在这个测试计划中，将发送两个 JDBC 请求。第一个是向 Eastman Kodak stock，第二个是向 Pfizer stock。同样，JMeter 按照它们在树中出现的顺序来发送请求。

(3) 编辑相关属性。

第一个属性：修改名称为Kodak，输入变量名MySQL(与在配置元件里面一样)，输入SQL查询字符串。第二个属性：修改名称为Pfizer，输入SQL查询字符串，如图 11-40 所示。

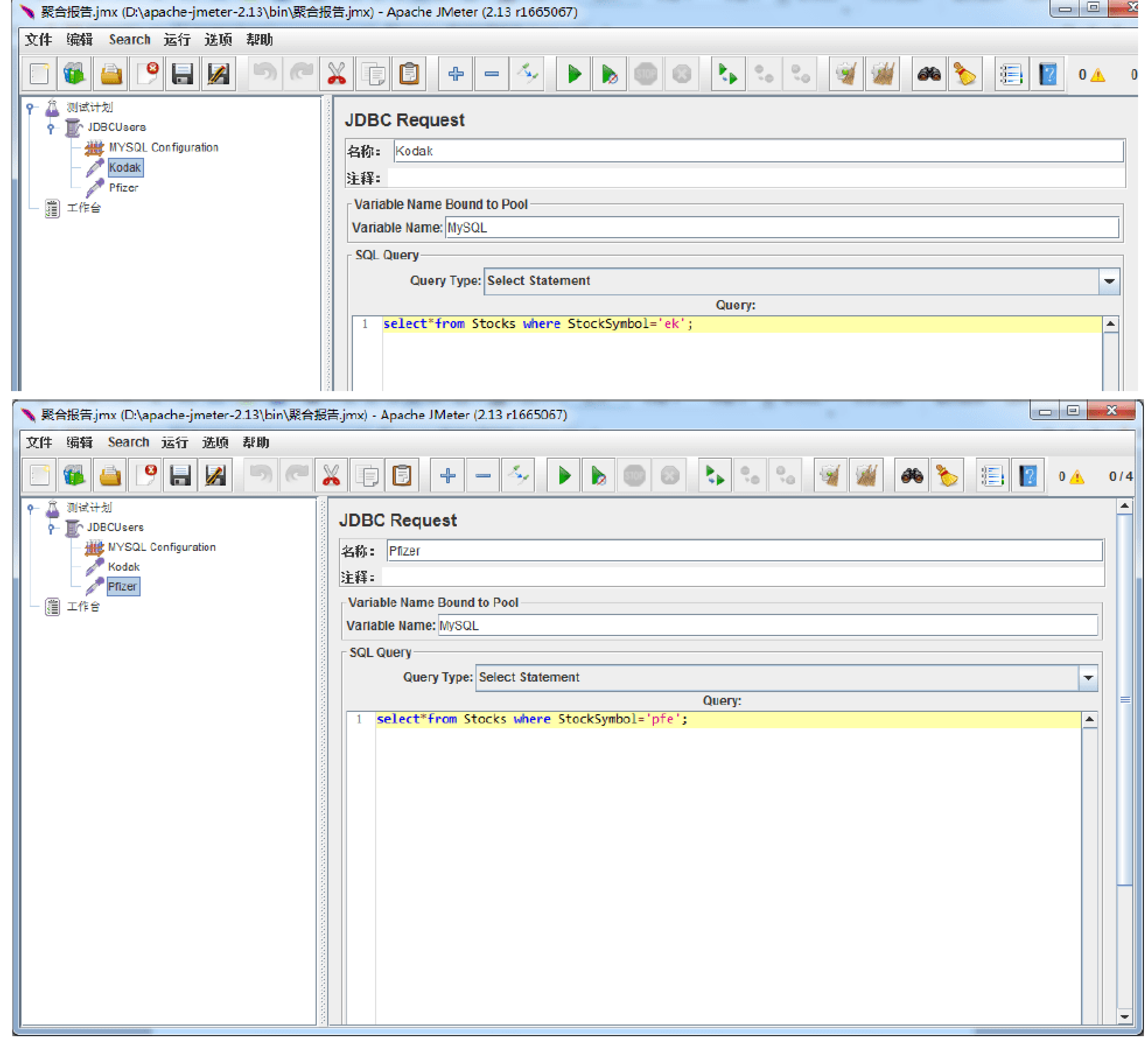


图 11-40　JDBC 请求设置

选择 JDBC 用户元件，添加一个图形结果监听器(“添加”|“监听器”|“图形结果”)，如图 11-41 所示。

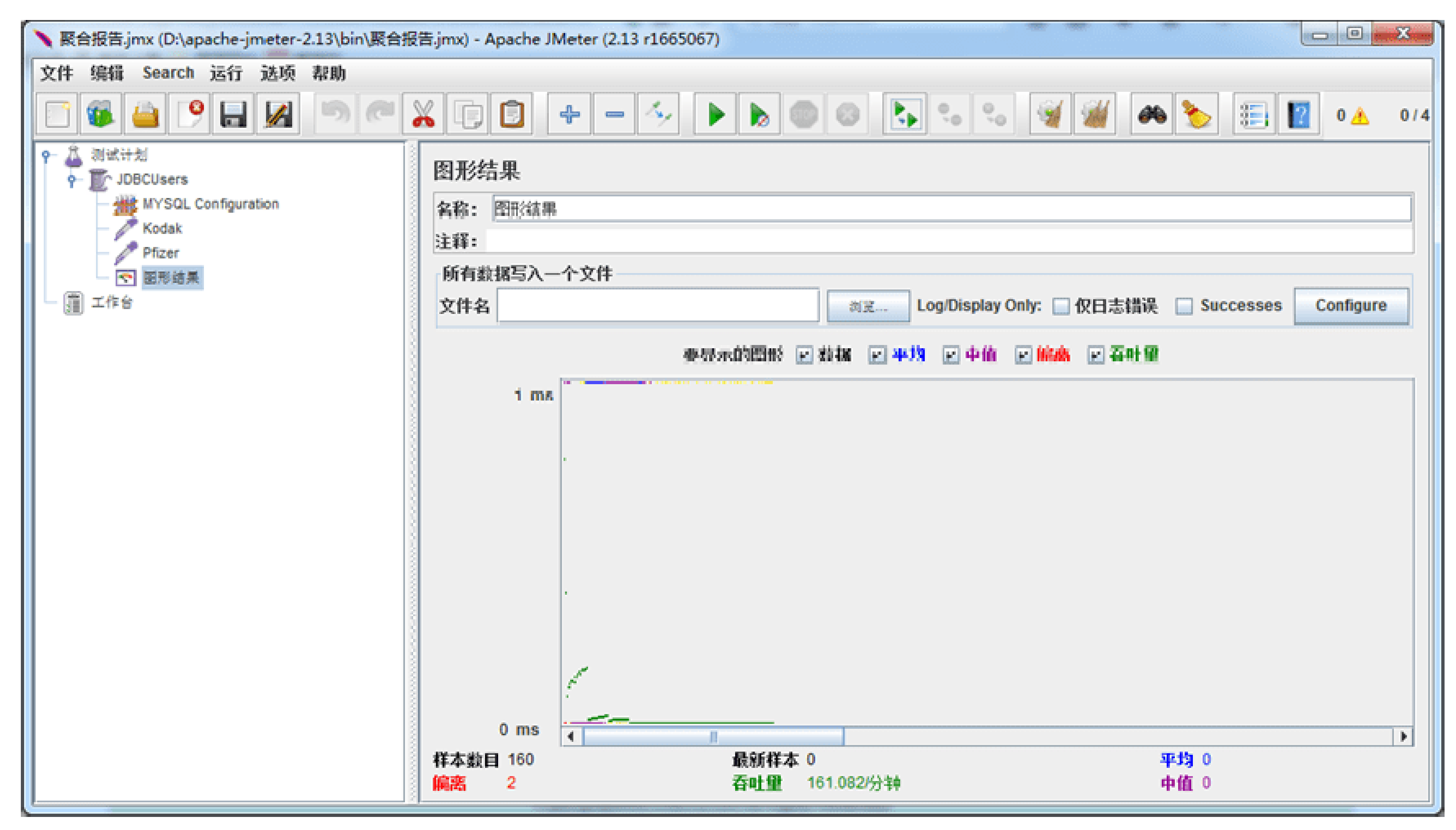


图 11-41　JDBC 图形监听结果

4) Web 应用测试

过程与 HTTP 测试基本相同，下面用不同的方式来查看性能测试结果(各种显示结果的含义可到 JMeter 官方网站阅读相关手册来查看)。

(1) 附带断言结果与图形结果如图 11-42 所示。

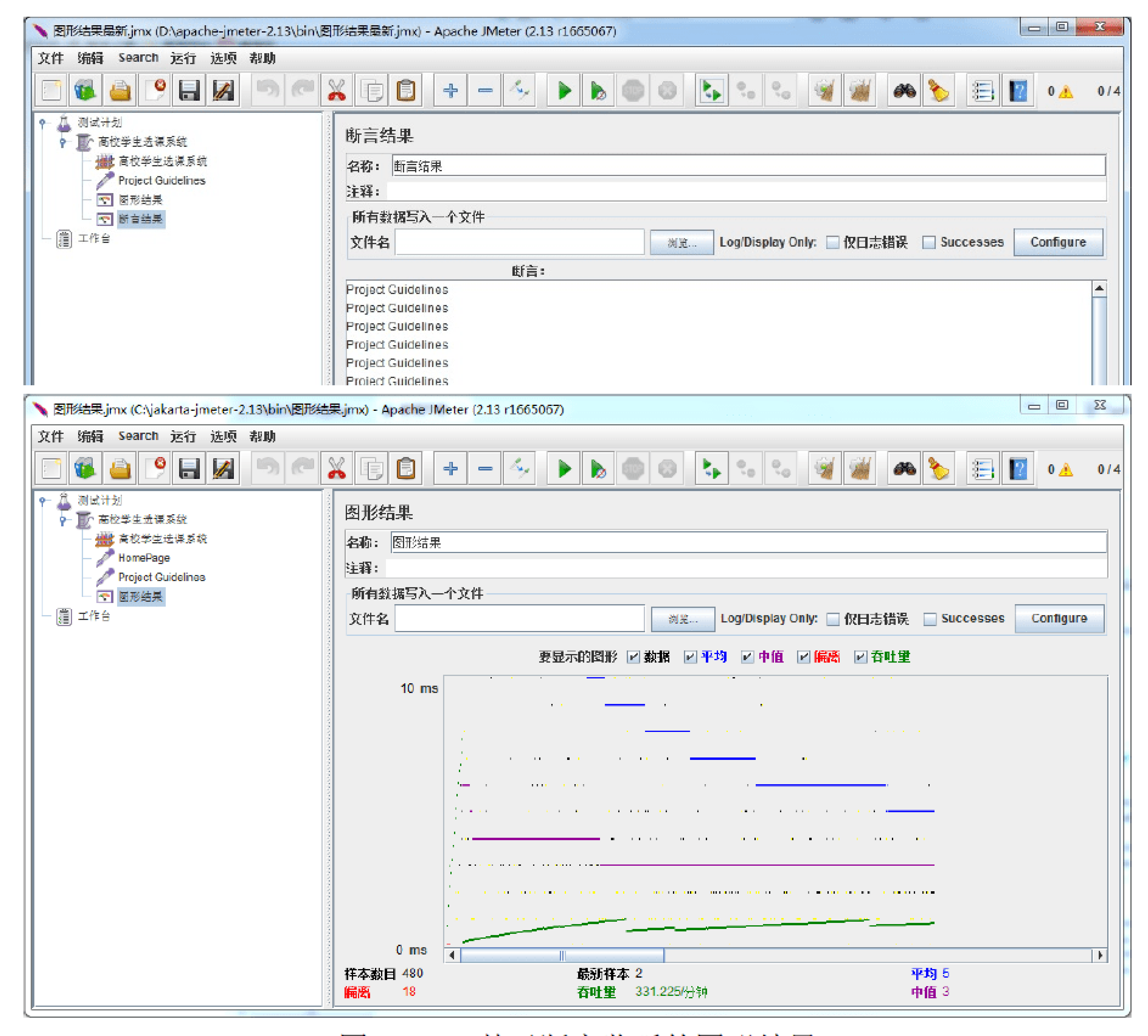


图 11-42　基于断言监听的图形结果

(2) 用表格查看结果及结果树，如图 11-43 与图 11-44 所示。

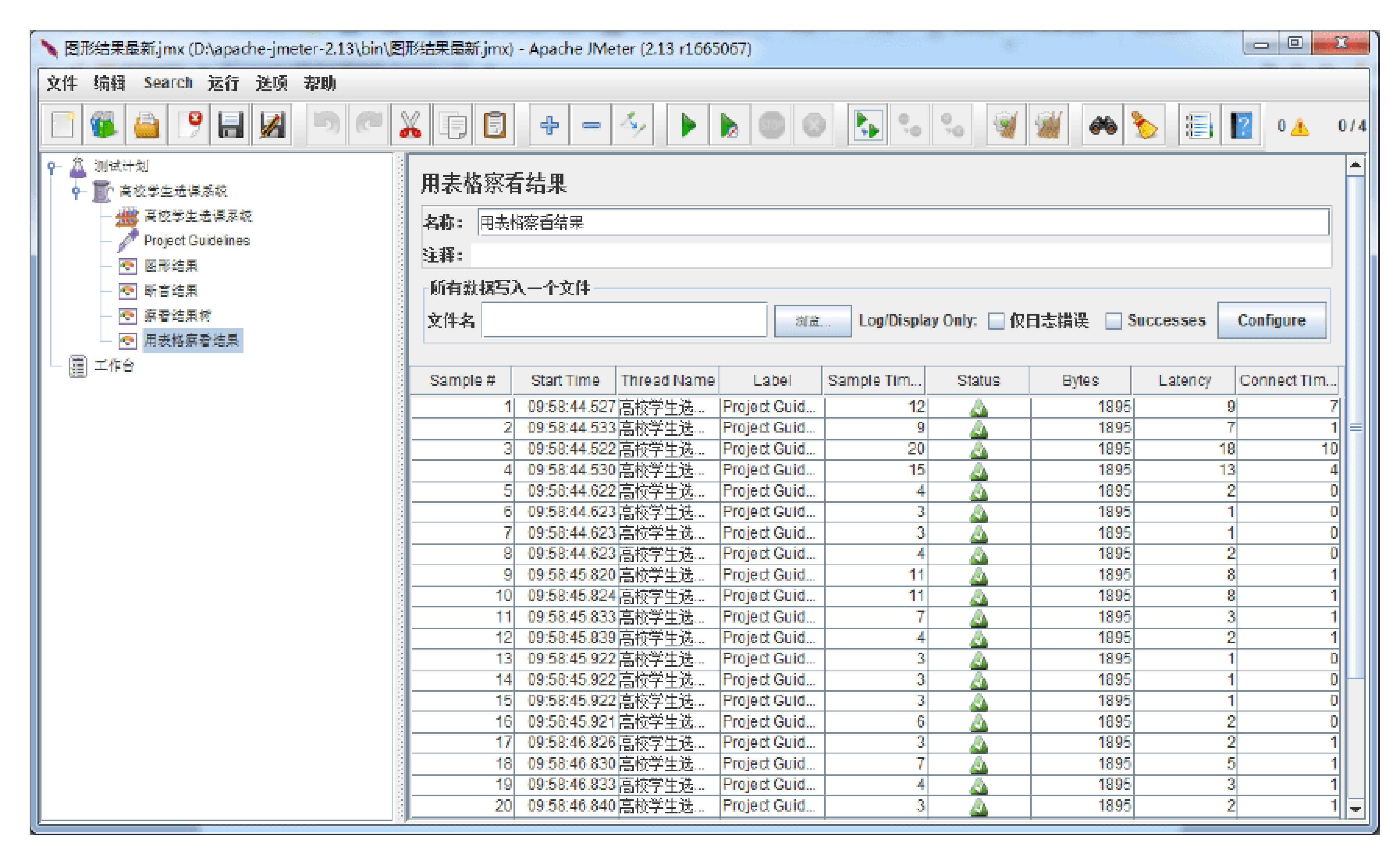


图 11-43　用表格查看结果

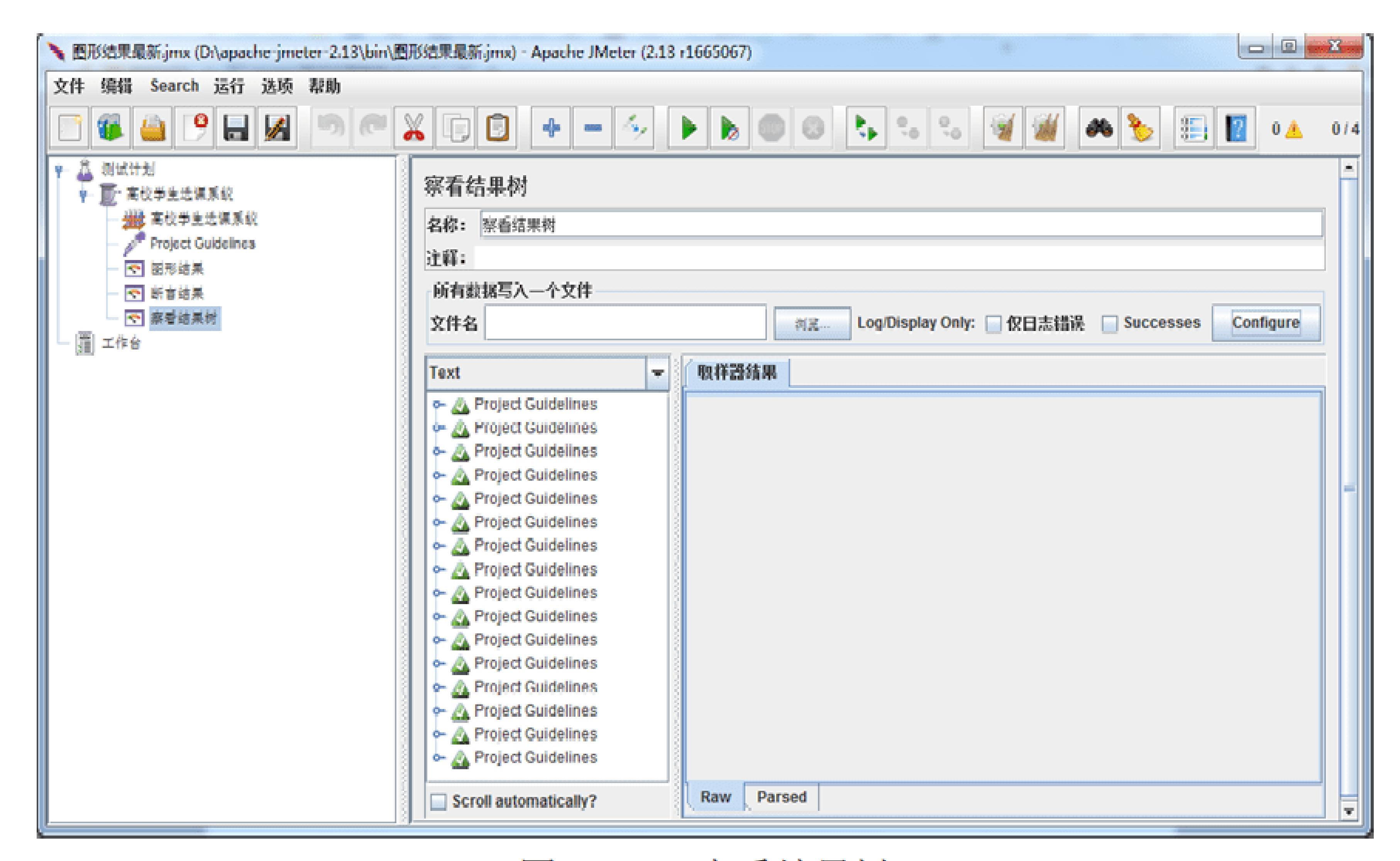
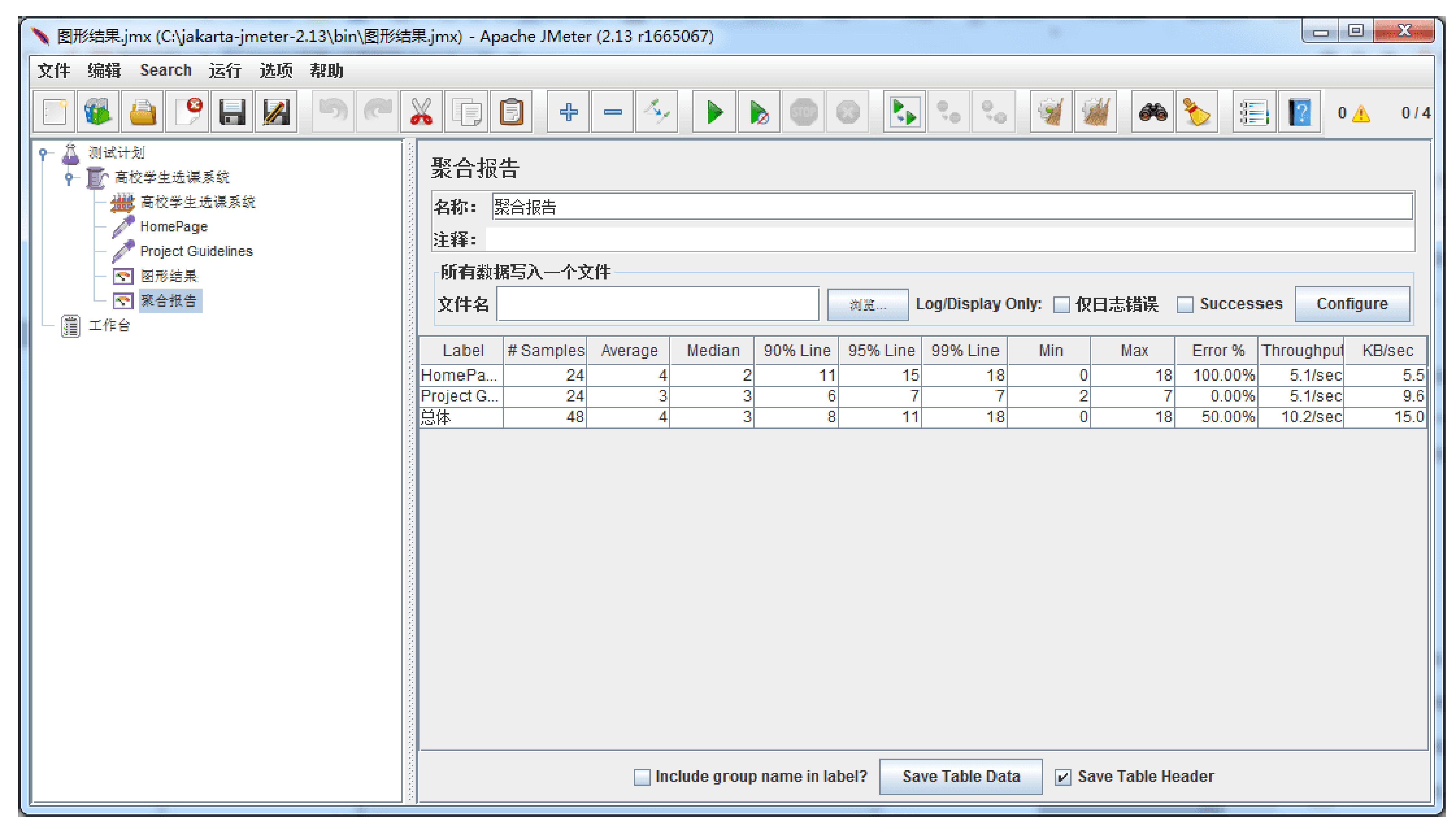


图 11-44　查看结果树

(3) 聚合报告结果如图 11-45 所示。

聚合报告标题解释：Label 用于说明请求类型，如 HTTP、FTP 等；#Samples 也就是图形报表中的样本数目，即总共发送到服务器的样本数目；Average 也就是图形报表中的平均值，它的值为总运行时间除以发送到服务器的请求数；Median 也就是图形报表中的中间值，是代表时间的数字，有一半的服务器响应时间低于该值，而另一半高于该值；90%Line 是指 90%请求的响应时间比得到的数值还小；Min 是代表时间的数字，是服务器响应的最短时间；Max 是代表时间的数字，是服务器响应的最长时间；Error%是请求的错误百分比；Throughput 也就是图形报表中的吞吐量，这里是指服务器每单位时间处理的请求数，注意查看是秒还是分；KB/sec 是每秒钟请求的字节数。



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	KB/sec
HomePa...	24	4	2	11	15	18	0	18	100.00%	5.1/sec	5.5
Project G...	24	3	3	6	7	7	2	7	0.00%	5.1/sec	9.6
总体	48	4	3	8	11	18	0	18	50.00%	10.2/sec	15.0

图 11-45　聚合报告结果

5) JMeter 工具小结

JMeter 是一款性能测试工具。与其说它是一个工具，不如说它是一个框架。因为 JMeter 的支持范围非常广，目前常见的需要进行性能测试的应用几乎都能应用(如文件、Servlet、Perl 脚本、Java 对象、数据库和查询、FTP 服务器等)。通过使用 JMeter 提供的功能，可以可视化地制订测试计划，包括规定使用什么样的负载、测试什么内容、传入的参数；同时，它还提供了多种图形化的测试结果显示方式，使用户能够更简便地开始测试工作和分析测试结果。JMeter 的一大好处就是它内部已经有实现好的线程机制，用户不用编写任何关于并发的东西，只需做简单配置即可。JMeter 还提供了一些类似插件的东西，用于线程运行时的控制。其次，JMeter 对测试结果能产生相应的统计报表，简单、直观，对于一般性能测试来说应该足够了。

习题和思考题

1. 什么是系统测试，简述系统测试的流程及各阶段任务。
2. 系统测试都有哪些要求？系统测试的主要内容有哪些？
3. 我们通常如何进行系统测试设计？测试用例设计及评估的方法有哪些？
4. 系统测试的主要手段有哪些？软件攻击和故障植入的方法有哪些？
5. 尝试用 Selenium 和 JMeter 对某一网站进行功能测试和系统测试。

第12章　面向对象软件测试

面向对象技术是现在很流行的软件开发技术，正逐渐代替之前被广泛使用的面向过程开发方法，被认为是解决软件危机的新兴技术。面向对象技术产生更好的系统结构，以及更规范的编程风格，极大地优化了数据使用的安全性，提高了程序代码的可重用性，一些人就此认为随着 OOA(Object-Oriented Analysis，面向对象分析)和 OOD(Object-Oriented Design，面向对象设计)的成熟，更多的设计模式重用将减轻面向对象系统的繁重测试量。应该看到，尽管面向对象技术的基本思想是要保证软件质量，但实际情况并非如此。因为无论采用什么样的编程技术，编程人员犯错误都是不可避免的，而且由于用面向对象技术开发的软件代码重用率高，因而就更需要严格测试，避免错误繁衍。被重用代码在新的环境中使用前，都要谨慎地重新测试。为了获得面向对象系统的高可靠性，可能需要更多而不是更少的测试。因此，软件测试并没有因面向对象编程的兴起而丧失它的重要性。

传统的软件测试策略是从小型测试开始，逐步走向大型测试。即从单元测试开始，然后逐步进入集成测试，最后是有效性测试和系统测试。在传统的测试中，单元测试集中在最小的可编译程序单位——子程序(如模块、子程序、进程)，一旦这些单元均被独立测试后，它们就被集成到程序结构中，这时要进行一系列的回归测试以发现由于模块接口带来的错误和新单元加入导致的负面作用。最后，系统被作为整体进行测试以保证发现需求中的错误。

面向对象技术独有的多态、继承、封装等新特性，导致传统程序设计不会产生的错误出现的可能性。例如，类的封装机制就给软件测试带来了困难。它把数据和操作数据的方法封装在一起，限制对象属性对外的可见性和外界对它的操作权限，这虽然有效地避免了类中有关实现细节的信息的使用，但这样的细节性信息也正是软件测试所不可忽略的，所以必须对面向对象软件测试进行研究。

从 1982 年在美国北卡罗来纳大学召开首次软件测试的正式技术会议至今，软件测试理论迅速发展，并相应出现了各种软件测试方法，使软件测试技术得到极大提高。然而，一度实践证明行之有效的软件测试对面向对象技术开发的软件多少显得有些力不从心。尤其是面向对象技术所独有的多态、继承、封装等新特性，产生了传统程序设计不会出现的新错误的可能性，或者使得传统软件测试中的重点不再显得突出，或者使原来测试经验认为和实践证明的次要方面成了主要问题。例如，在传统的面向过程程序中，对于函数：

```
y=Function(x);
```

只需要考虑一个函数(Function( ))的行为特点，而在面向对象程序中，我们不得不同时考虑基类函数(Base::Function( ))的行为和继承类函数(Derived::Function( ))的行为。

面向对象程序的结构不再是传统的功能模块结构，由于面向对象软件的封装性导致没有传统结构化程序的层次式控制结构，而是作为整体，因而原有集成测试所要求的逐步将

开发的模块搭建在一起进行测试的方法已成为不可能。而且面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。单元测试失去了本身的多数意义，传统的自顶向下和自底向上的集成测试策略也有了较大改变。

因此，传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点，应该有一种新的测试模型。

12.1　面向对象程序设计语言对软件测试的影响

面向对象程序设计语言的出现不仅改变了程序设计的风格，而且还影响了软件开发的全过程。面向对象的软件需求分析方法、设计方法也应运而生。然而，面向对象方法对软件测试的影响直至近年来才开始为人们所注意。下面我们首先分析面向对象程序设计语言的特征对软件测试的影响。

12.1.1　信息隐蔽对测试的影响

类的重要作用之一是信息隐蔽。它对类中所封装信息的连接进行控制，从而避免类中有关实现细节的信息被错误地使用，而这样的细节性信息正是软件测试所不可忽略的。由于面向对象的软件系统在运行时刻由一组协调工作的对象组成，对象具有一定的状态，因此对于面向对象的程序测试来说，对象的状态是必须考虑的因素，测试应涉及对象的初态、输入参数、输出参数、对象的终态。对象的行为是被动的，它只有在接收有关信件后才被激活来进行所请求的工作，并将结果返回给发信者。在工作过程中对象的状态可能被修改，产生新的状态，面向对象软件测试的基本工作就是创建对象(包括初始化)，面向对象发送一系列信息，然后检查结果对象的状态，看其是否处于正确的状态。问题是对象的状态往往是隐蔽的，若类中未提供足够的存取函数来表明对象的实现方式和内部状态，则测试者必须增添这样的函数。因此，类的信息隐蔽机制给测试带来困难。

12.1.2　封装和继承对测试的影响

在面向对象的程序中，由于继承的作用，一个函数可能被封装在多个类中，子类中还可以对继承的特征进行重定义。问题是，未重定义的继承特征是否还需要进行测试呢？重定义的特征需要重新测试是显然的，但如何测试重定义的特征呢？E.J.Weyuker 曾经提出了 11 条基于程序的测试数据集的充分性公理，D.E.Perry 与 G.E.Kaiser 根据 Weyuker 公理对这些问题进行了讨论，结论是：封装和继承并未简化测试问题，反而使测试更加复杂。

12.1.3　集成测试

对于由传统程序设计语言编写的软件，软件测试人员普遍接受三个级别的测试：单元测试、集成测试和系统测试。一个单元可以是一个函数或一个模块，虽然单元的定义可能不同，但单元测试的目的都是通过测试残桩(stub)和驱动程序来模拟和该单元相关的其他单元，以验证该单元自身是否正确地工作。当各单元被分别测试后，集成测试根据设计阶段

形成的功能分解树，自顶向下或自底向上逐步用各个单元替换残桩和驱动程序。集成测试强调软件的结构和界面，测试往往取决于存储的状态。而系统测试强调软件的行为，系统测试者从用户的角度，观察系统级的输入和输出，以确定系统的行为是否与需求定义一致。无论在哪个级别上进行测试，测试过程均为输入测试数据、处理、验证输出结果这三个步骤。

对面向对象的程序测试应当分为多少级别尚未达成共识。一般认为，面向对象的程序也和其他语言的程序一样，都要进行系统级测试。M.D.Smith 和 D.J.Robson 从面向对象程序的结构出发，认为面向对象的程序测试应当分为 4 个级别：①方法级，考查一个方法对数据进行的操作；②类级，考查封装在一个类中的方法和数据之间的相互作用；③簇级，考查一组协同操作的类之间的相互作用；④系统级，考查由所有类和主程序构成的整个系统。

他们认为，上述 4 个级别中的方法级测试和传统的单元测试相对应。也有人认为在面向对象程序中，最小的可测试单元已不是方法，而是类和类的实例。应该说，这些看法和传统的单元测试中对单元的认识还是一致的。已经有经验表明类级测试是必需的，也是发现错误的重要手段，以簇作为测试的最小单元会导致某些应当在开发周期中较早发现的错误延至系统测试时才发现。但传统的集成测试和面向对象的程序测试中的对应尚没有一致的看法。显然，基于功能分解的自顶向下和自底向上的集成测试策略并不适用于用面向对象方法构造的软件。在用各个方法分别测试之后，每次任选一个方法集成到类中，逐步进行测试，直至形成一个完整的类，这种集成策略也未必合适，原因是各个方法之间可能有相互作用，某一方法可能要求对象处于某个特定的状态，而该状态必须由其他方法设置，所以还需要考虑集成的次序问题。

基于结构的传统集成策略并不适于面向对象的程序。这是因为面向对象的程序的执行实际上是执行一个由信息连接起来的方法序列，而这个方法序列往往是由外部事件驱动的。根据这一执行特性，P.C.Jorgensen 和 C.Erickson 认为，面向对象的测试应该分为 5 个层次：①方法测试；②方法/信息路径(MMPath)测试，即执行一个方法/信息序列，直至到达一个不再发送信息的方法；③系统基本功能测试，即测试从一个输入端口事件开始，由此触发一条方法/信息路径的执行，最后终结于某一输出端口事件；④线程测试；⑤线程间相互作用测试。其中方法测试对应于单元测试，方法/信息路径测试和系统基本功能测试对应于集成测试，而线程测试和线程间相互作用测试对应于系统测试。

12.1.4　多态性和动态绑定对测试的影响

多态性和动态绑定为程序的执行带来了不确定性，给软件测试带来了新的挑战。多态性和动态绑定是面向对象方法的关键特性之一。同一消息可以根据发送消息对象的不同采用多种不同的行为方式，这就是多态的概念。比如根据当前指针引用的对象类型来决定使用正确的方法，这就是多态性的行为操作。运行时系统能自动为给定消息选择合适的实现代码，这给程序员提供了高度柔性、问题抽象和易于维护。

例如：假设有类 A、类 B 和类 C 三个类，类 B 继承类 A，类 C 又继承类 B。成员函数 a( )分别存在于这三个类中，但在每个类中的具体实现则不同。同时在程序中存在一个函数 fn( )，该函数在形参中创建了类 A 的一个实例 Ca，并在函数中调用了方法 a( )。程序

运行时相当于执行了一条分支语句 switch，首先判定传递过来的实参的类型(类 A、类 B 或类 C)，然后确定究竟执行哪一个类中的方法 a( )。

在测试时必须为每一个分支生成测试用例，以覆盖所有的分支和所有的程序代码。因而，多态性和动态绑定所带来的不确定性，使得传统测试实践中的静态分析方法遇到了不可逾越的障碍，也增加了系统运行中可能的执行路径，加大了测试用例选取的难度和数量。多态性给软件测试带来的问题仍然是目前研究的重点及难点问题之一。

从上述分析可知，虽然信息隐蔽与封装使得类具有较好的相对独立性，有利于提高软件的易测试性和保证软件的质量，但是，这些机制与继承机制和动态绑定给软件测试带来了新的难题。尤其是面向对象软件中类与类之间的集成测试和类中各个方法之间的集成测试具有特别重要的意义，与用传统语言编写的软件相比，集成测试的方法和策略也应该有所不同。

12.2　面向对象测试模型

面向对象软件的开发是从分析和设计模型的创建开始的。模型从对系统需求相对非正式的表示开始，逐步演化为详细的类模型、类连接和关系、系统设计和分配，以及对象设计在每个阶段的测试模型。面向对象分析的测试包括对认定的对象的测试、对认定的结构的测试、对认定的主题的测试、对定义的属性和实例关联的测试、对定义的服务和消息关联的测试等内容。面向对象设计的测试包括对认定的类的测试、对构造的类层次结构的测试、对类库支持的测试等内容。面向对象编程的测试主要是类测试，主要测试数据成员是否满足数据封装的要求以及类是否实现了要求的功能。

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。针对这种开发模型，结合传统的测试步骤的划分，这里建议一种整个软件开发过程中不断测试的测试模型，使开发阶段的测试与编码完成后的单元测试、集成测试、系统测试成为一个整体(参见表 12-1)。测试模型如图 12-1 所示。

表 12-1　面向对象开发与测试

OO：面向对象	OOA：面向对象分析
OOD：面向对象设计	OOP：面向对象编程
OOA Test：面向对象分析的测试	OOD Test：面向对象设计的测试
OOP Test：面向对象编程的测试	OO Unit Test 面向对象的单元测试
OO Integrate Test：面向对象的集成测试	OO System Test：面向对象的系统测试

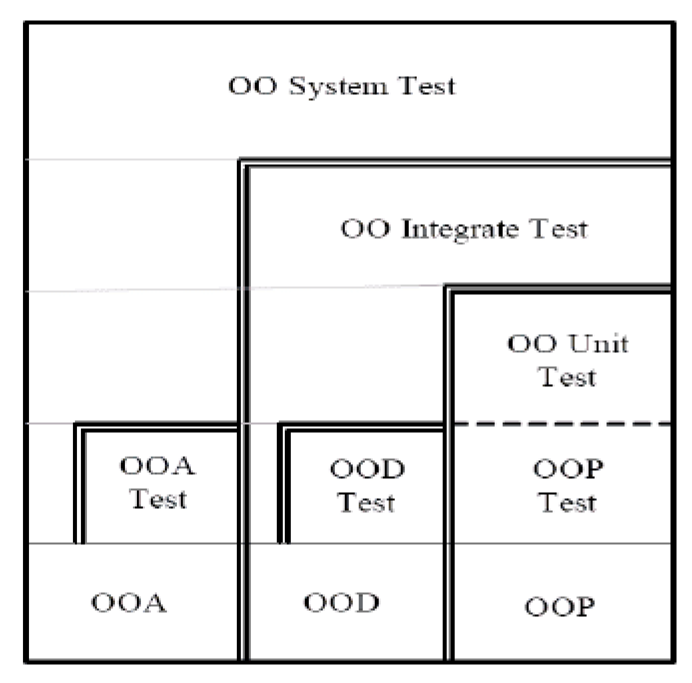


图 12-1　面向对象软件测试模型

OOA Test 和 OOD Test 是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代码实现进行测试，主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 C++语言实现的，那么主要就是对类的成员函数进行测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用、类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见 OOD 或 OOD Test 结果(详见后面内容)。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴 OOA 或 OOA Test 结果。

尽管上述各阶段的测试构成了一个相互作用的整体，但测试的主体、方向和方法各有不同，为了叙述方便，我们接下来将从 OOA 测试、OOD 测试、OOP 测试、单元测试、集成测试、系统测试 6 个方面分别介绍面向对象软件测试。

12.2.1　面向对象分析的测试(OOA Test)

传统的面向过程分析是一个功能分解的过程，是把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析(OOA)是把 E-R 图和语义网络模型，即信息造型中的概念，与面向对象程序设计语言中的重要概念结合在一起形成的分析方法，最后通常得到问题空间的图表形式的描述。

OOA 直接映射问题空间，全面地对问题空间中的解进行现实抽象化。将问题空间中的实例抽象为对象(不同于 C++中的对象概念)，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和服务表示实例的特性和行为。对于一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性。OOA 的结果是为后面阶段类的选定和实现，以及类层次结构的组织和实现提供平台。因此，OOA 对问题空间分析抽象的不完整，最终会影响软件的功能实现，导致软件开发后期大量可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。因此，我们对 OOA 的测试重点在于其完整性和冗余性。

尽管 OOA 的测试是一个不可分割的系统过程，但是为了叙述方便，我们将 OOA 阶段的测试划分为 5 个方面：①对认定的对象的测试；②对认定的结构的测试；③对认定的主题的测试；④对定义的属性和实例关联的测试；⑤对定义的服务和消息关联的测试。

对象、结构、主题等在 OOA 结果中的位置，我们可以参考图 12-2。

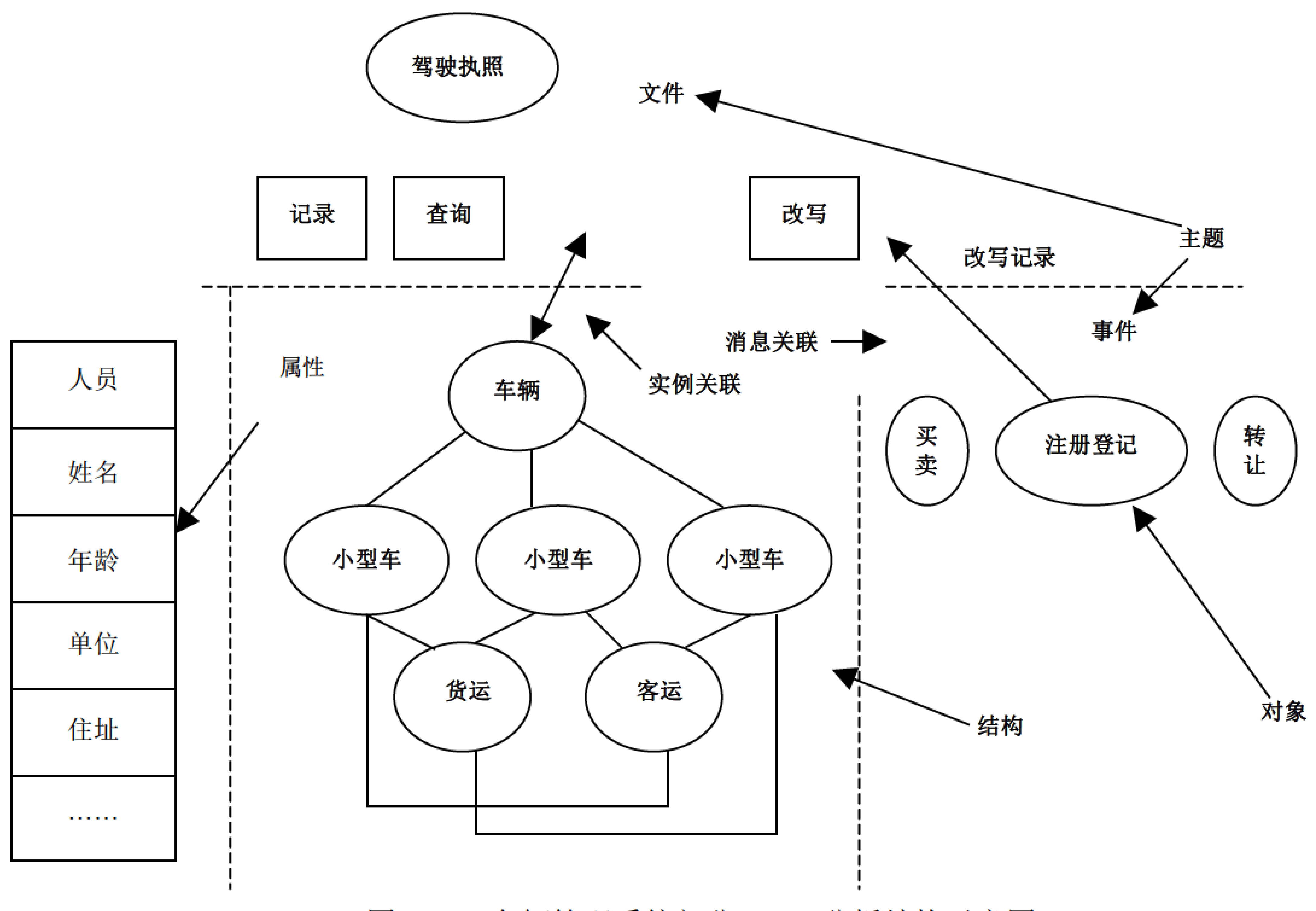


图 12-2　车辆管理系统部分 OOA 分析结构示意图

1. 对认定的对象的测试

OOA 中认定的对象是对问题空间中结构、其他系统、设备、被记忆的事件、系统设计人员等实际实例的抽象。对它的测试可以从这方面考虑：①认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对象中；②认定的对象是否具有多个属性(只有一个属性的对象通常应看成其他对象的属性，而不是抽象为独立的对象)；③对认定为同一对象的实例是否有共同的区别于其他实例的共同属性；④对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示；⑤如果系统没有必要始终保持对象代表的实例的信息，提供或得到关于它的服务，则认定的对象也无必要去关注；⑥认定的对象的名称应该尽量准确、适用。

2. 对认定的结构的测试

认定的结构指的是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊的关系，组装结构体现了问题空间中实例的整体与局部的关系。

对认定的分类结构的测试可从这几方面着手：①对于结构中的一种对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象；②对于结构中的一种对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象；③对于所有认定的对象，是否能在问题空间中向上层抽象出在现实中有意义的对象；④高层对象的特性是否完全体现下层对象的共性；⑤低层的对象是否有高层特性基础上的特殊性。

对认定的组装结构的测试可从这几方面入手：①整体(对象)和部件(对象)的组装关系是否符合现实的关系；②整体(对象)的部件(对象)是否在考虑的问题空间中有实际应用；③整体(对象)中是否遗漏了反映问题空间中有用的部件(对象)；④部件(对象)是否能够在问题空间中组装新的有现实意义的整体(对象)。

3. 对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象，是为了提供 OOA 分析结构的可见性，如同文章对各部分内容的概要。对主题层的测试应该考虑：①贯彻 George Miller 的 7+2 原则，如果主题个数超过 7 个，就要求对有较密切属性和服务的主题进行归并；②主题所反映的一组对象和结构是否具有相同和相近的属性和服务；③认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA 结果的概貌(尤其是对于非技术人员的 OOA 结果读者而言)；④主题间的消息联系(抽象)是否代表了主题所反映的对象和结构之间的所有关联。

4. 对定义的属性和实例关联的测试

属性用来描述对象或结构所反映的实例的特性，而实例关联用来反映实例集合间的映射关系。对属性和实例关联的测试可从这几方面考虑：①定义的属性是否对相应的对象和分类结构的每个现实实例都适用；②定义的属性在现实世界中是否与这种实例关系密切；③定义的属性是否能够不依赖于其他属性被独立理解；④定义的属性在问题空间中是否与这种实例关系密切；⑤定义的属性在分类结构中的位置是否恰当，底层对象的共有属性是否在上层对象属性中体现；⑥问题空间中每个对象的属性是否定义完整；⑦定义的实例关联是否符合现实；⑧问题空间中的实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

5. 对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间中所要求的行为。在问题空间中，由于实例间需要通信，因此在 OOA 中需要为它们定义消息关联。对定义的服务和消息关联的测试可从这几方面进行：①对象和结构在问题空间中的不同状态是否定义了相应的服务；②对象或结构所需要的服务是否都定义了相应的消息关联；③定义的消息关联所指引的服务提供是否正确；④沿着消息关联执行的线程是否合理，是否符合现实过程；⑤定义的服务是否重复，是否定义了能够得到的服务。

12.2.2　面向对象设计的测试(OOD Test)

通常的结构化设计方法，用的是面向作业的设计方法，它把系统分解以后，提出一组作业，这些作业是以过程实现系统的基础构造，把问题域的分析转换为求解域的设计，分析的结果是设计阶段的输入。

而面向对象设计(OOD)采用造型的观点，以 OOA 为基础归纳出类，并建立类结构或进一步构造类库，实现分析结果对问题空间的抽象。OOD 归纳的类，可以是对象简单的延续，可以是不同对象的相同或相似的服务。由此可见，OOD 不是 OOA 上的另一思维方式的大动干戈，而是对 OOA 的进一步细化和抽象。所以，OOD 与 OOA 的界限通常难以严格区分。OOD 确定类和类结构不仅满足当前需求分析的要求，更重要的是通过重新组合或

加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。因此，对 OOD 的测试，建议针对功能的实现和重用，以及对 OOA 结果的拓展，从三方面考虑：①对认定的类的测试；②对构造的类层次结构的测试；③对类库的支持的测试。

1. 对认定的类的测试

OOD 认定的类可以是 OOA 中认定的对象，也可以是对象所需要服务的抽象，以及对象所具有属性的抽象。认定的类原则上应该具备基础性，这样才便于维护和重用。测试认定的类：①是否涵盖了 OOA 中所有认定的对象；②是否能体现 OOA 中定义的属性；③是否能实现 OOA 中定义的服务；④是否对应一个含义明确的数据抽象；⑤是否尽可能少地依赖其他类；⑥类中的方法(C++：类的成员函数)是否为单用途。

2. 对构造的类层次结构的测试

为了能充分发挥面向对象的继承共享特性，OOD 的类层次结构通常基于 OOA 中产生的分类结构的原则来组织，着重体现父类和子类间的一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，测试这几方面：①类层次结构是否涵盖了所有定义的类；②是否能体现 OOA 中所定义的实例关联；③是否能实现 OOA 中所定义的消息关联；④子类是否具有父类中没有的新特性；⑤子类间的共同特性是否完全在父类中得以体现。

3. 对类库支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是再次软件开发的重用。由于它并不直接影响当前软件的开发和功能实现，因此将其单独提出来测试，也可作为对高质量类层次结构的评估。拟定的测试点有：①一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口(包括名字和参数表)；②类中的方法(C++：类的成员函数)功能是否较单纯，相应的代码行是否较少(建议不超过 30 行)；③类的层次结构是否是深度大、宽度小。

12.2.3　面向对象编程的测试(OOP Test)

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率提高。继承使得传统测试遇见这样一个难题：对继承的代码究竟应该怎样测试(参见面向对象单元测试)？多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内同一函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误精确地确定在某一具体的类。因此，在面向对象编程(OOP)阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现在两个方面(假设编程使用 C++语言)：①数据成员是否满足数据封装的要求；②类是否实现了要求的功能。

1. 数据成员是否满足数据封装的要求

数据封装是数据和数据有关的操作的集合。检查数据成员是否满足数据封装的要求，基本原则是数据成员是否被外界(数据成员所属的类或子类以外的调用)直接调用。更直观地说，当改变数据成员的结构时，是否影响了类的对外接口，是否会导致相应外界必须改动。值得注意的是，有时强制的类型转换会破坏数据的封装特性。例如：

```
Class Hiden
{
Private:
    int    a=1;
    char   *p=&quot;hidden&quot;;
}
Class Visible
{
Public:
   int   b=2;
   char *s=&quot;visible&quot;;
}
…
…
Hiden pp;
Visible *qq=(Visible *)&amp;#38;pp;
```

在上面的程序段中，pp 的数据成员可以通过 qq 被随意访问。

2. 类是否实现了要求的功能

类所实现的功能，都通过类的成员函数来执行。在测试类的功能实现时，应该首先保证类成员函数的正确性。单独地看待类的成员函数，与面向过程程序中的函数或过程没有本质上的区别，几乎所有传统的单元测试中所使用的方法，都可在面向对象的单元测试中使用(具体的测试方法会在面向对象的单元测试部分介绍)。类函数成员的正确行为只是类能够实现要求功能的基础，类成员函数间的作用和类之间的服务调用是单元测试无法确定的。因此，需要进行面向对象的集成测试(具体的测试方法会在面向对象的集成测试部分介绍)。需要着重声明，测试类的功能，不能仅满足于代码能无错运行或被测试类能提供的功能无错，应该以所做的 OOD 结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。必要时(如通过 OOD 结果仍不清楚、明确的地方)还应该参照 OOA 的结果，以之为最终标准。

12.2.4　面向对象的单元测试(OO Unit Test)

传统的单元测试针对程序的函数、过程或完成一定功能的程序块。沿用单元测试的概念，面向对象的单元测试实际测试类的成员函数。一些传统的测试方法在面向对象的单元测试中都可以使用，如等价类划分法、因果图法、边界值分析法、逻辑覆盖法、路径分析法、程序插装法等。

面向对象软件的单元概念发生了变化。封装驱动了类和对象的定义，这意味着每个类和类的实例(对象)包装了属性(数据)和操纵这些数据的操作(也称为方法或服务)，而不是个

体的模块。最小的可测试单位是封装的类或对象，类包含一组不同的操作，并且某特殊操作可能作为一组不同类的一部分存在。因此，单元测试的意义发生了较大变化。

我们不再孤立地测试单个操作(传统的单元测试观点)，而是将操作作为类的一部分。对 OO 软件的类测试等价于传统软件的单元测试。和传统软件的单元测试不一样，它往往关注模块的算法细节和模块接口间流动的数据，OO 软件的类测试是由封装在类中的操作和类的状态行为驱动的。

用于单元级测试的测试分析(提出相应的测试要求)和测试用例(选择适当的输入，达到测试要求)的规模和难度等，远小于后面将介绍的对整个系统的测试分析和测试用例，而且强调对语句应该有 100%的执行代码覆盖率。在设计测试用例、选择输入数据时，可以基于这两个假设：①如果函数(程序)对某一类输入中的一个数据正确执行，则对同类中的其他输入也能正确执行；②如果函数(程序)对某一复杂度的输入正确执行，则对更高复杂度的输入也能正确执行。例如需要选择字符串作为输入时，基于本假设，就无须计较字符串的长度。除非字符串的长度是要求固定的，如 IP 地址字符串。

在面向对象程序中，类的成员函数通常都很小，功能单一，函数间调用频繁，容易出现一些不宜发现的错误。例如：

```
if ( -1 == write(fid， buffer ,amount)) error_out();
```

该语句没有全面检查 write( )的返回值，无意中断然假设了只有数据被完全写入和没有写入两种情况。如果测试也忽略了数据部分写入的情况，就会给程序遗留隐患：①按程序的设计，使用函数 strrchr( )查找最后的匹配字符，但程序中误写成了函数 strchr( )，使程序功能实现时查找的是第一个匹配字符；②程序中将 if(strncmp(str1,str2,strlen(str1)))误写成了 if(strncmp(str1,str2,strlen(str2)))，如果测试用例中使用的数据 str1 和 str2 长度一样，就无法检测出。

因此，在做测试分析和设计测试用例时，应注意面向对象程序的这个特点，仔细地进行测试分析和设计测试用例，尤其是针对以函数返回值作为条件判断选择字符串操作的情况。

面向对象编程的特性使得对成员函数的测试，又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性，使子类继承或重载的父类成员函数出现了传统测试中未曾遇见的问题。Brian Marick 给出了两方面的考虑：

1. 继承的成员函数是否都不需要测试

对于父类中已经测试过的成员函数，下面两种情况需要在子类中重新测试：继承的成员函数在子类中做了改动；成员函数调用了改动过的成员函数的部分。例如：假设父类 Bass 有两个成员函数 Inherited( )和 Redefined( )，子类 Derived 只对 Redefined( )做了改动。Derived::Redefined( )显然需要重新测试。对于 Derived::Inherited( )，如果它有调用 Redefined( )的语句(如 x=x/Redefined( ))，就需要重新测试，反之，无此必要。

2. 对父类的测试是否能照搬到子类

延用上面的假设，Base::Redefined( )和 Derived::Redefined( )已经是不同的成员函数，它们有不同的服务说明和执行。对此，照理应该对 Derived::Redefined( )重新测试分析，设

计测试用例。但由于面向对象的继承使得两个函数又相似，因此只需要在 Base::Redefined( ) 的测试要求和测试用例上添加对 Derived::Redfined( )新的测试要求和增补相应的测试用例即可。

例如：Base::Redefined( )含有如下语句

```
if (value & lt;0) message (&quot ; less & quot;);
else if (value == 0) message (&quot ; equal & quot;);
else message (&quot ; more & quot;);
```

Derived: Redfined( )中定义为：

```
if (value & lt ; 0) message (&quot ;less & quot;);
else if (value == 0) message (&quot ; It is equal &quot;);
else
{
    Message (&quot ; more & quot;);
    if (value == 88) message (&quot ; luck & quot;);
}
```

在原有的测试上，只需要对 Derived::Redfined( )的测试做如下改动：改动 value== 0 的测试期望结果；增加 value==88 的测试。

多态有几种不同的形式，如参数多态、包含多态、重载多态。包含多态和重载多态在面向对象语言中通常体现在子类与父类的继承关系上，对这两种多态的测试参见上述对父类成员函数继承和重载的论述。包含多态虽然使成员函数的参数可有多种类型，但通常只是增加了测试的复杂度。对具有包含多态的成员函数测试时，只需要在原有的测试分析和基础上扩大测试用例中输入数据的类型考虑。

12.2.5　面向对象的集成测试(OO Integrate Test)

传统的集成测试，是对由底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序，相互调用的功能散布在程序的不同类中，类通过消息相互作用来申请和提供服务。类的行为与其状态密切相关，状态不仅仅体现类数据成员的值，也许还包括其他类中的状态信息。由此可见，类相互依赖、极其紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于黑盒的集成测试。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的一些测试软件都提供了程序理解的功能，即通过原程序得到类关系图和函数功能调用关系图。将程序理解得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测 OOP 是否达到了设计要求。

为动态测试设计测试用例时，通常需要以上述功能调用结构图、类关系图或实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

具体设计测试用例时，可参考这几个步骤：①先选定检测的类，参考 OOD 分析结果，仔细检测出类的状态和相应的行为、类或成员函数间传递的消息、输入或输出的界定等；②确定覆盖标准；③利用结构关系图确定待测类的所有关联；④根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

值得注意的是，设计测试用例时，不但要设计确认类功能满足的输入，还应该有意识地设计一些被禁止的例子，确认类是否有不合法的行为产生，如发送与类状态不相适应的消息或与要求不相适应的服务等。根据具体情况，动态地集成测试，有时也可以通过系统测试来完成。

12.2.6　面向对象的系统测试(OO System Test)

通过单元测试和集成测试，仅能保证软件开发的功能得以实现，而不能确认在实际运行时是否满足用户的需要，是否大量存在实际使用条件下会被诱发产生错误的隐患。为此，对完成开发的软件必须进行规范的系统测试。换个角度说，开发完成的软件仅仅是实际投入使用系统的一个组成部分，需要测试它与系统其他部分配套运行的表现，以保证在系统各部分协调工作的环境下也能正常工作。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA 分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全再现问题空间。系统测试不仅仅检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。

系统测试是对所有类和主程序构成的整个系统进行整体测试，以验证软件系统的正确性和性能指标等满足需求规格说明书和任务书指定的要求。体现的具体测试内容包括：

(1) 功能测试：测试是否满足开发要求，是否能够提供设计所描述的功能，用户的需求是否都得到满足。功能测试是系统测试最常用和必需的测试，通常还会以正式的软件说明书为测试标准。

(2) 强度测试：测试系统的能力最高实际限度，即软件在一些超负荷情况下的功能实现情况，如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

(3) 性能测试：测试软件的运行性能。这种测试常常与强度测试结合进行，需要事先对被测软件提出性能指标，如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

(4) 安全测试：验证安装在系统中的保护机制确实能够对系统进行保护，使之不受各种非常的干扰。安全测试时需要设计一些测试用例，试图突破系统的安全保密措施，检验

系统是否存在安全保密漏洞。

(5) 恢复测试：采用人工的干扰使软件出错，中断使用，检测系统的恢复能力，特别是通信系统。恢复测试时，应该参考性能测试的相关测试指标。

(6) 可用性测试：测试用户是否能够满意使用。具体体现为操作是否方便，用户界面是否友好等。

(7) 安装/卸载测试(install/uninstall test)，等等。

系统测试需要对被测软件结合需求分析做仔细的测试分析，建立测试用例。测试用例可从对象-行为模型和作为面向对象分析的事件流图中导出。

12.2.7　面向对象软件的回归测试

面向对象软件的回归测试不再作为测试的一个独立阶段，而是以增量方式进行的。对象系统中的交互，既发生在类内的方法间，又发生在多个类之间。比如类 A 与类 B 的交互可通过：①将类 B 的实例变量作为参数传给类 A 的某方法。这时，类 B 的改变必然导致对类 A 的方法的回归测试。如果类 A 的方法又与其他方法发生交互，这些方法及其类就必须都进行回归测试；②类 A 的实例是类 B 表示的一部分。这时，类 B 的所有对类 A 中变量进行引用的方法都必须进行回归测试。

在这两种情况下，测试依赖集可用来确定第一种情况中所有与类 A 的交互集；对象的数据依赖集则可用来确定上述两种情况中所有必须重新测试的类和方法。

对面向对象软件的修改主要有：①不修改对象属性，只对方法进行修改。该对象以及系统中所有与该对象发生交互的对象都要进行完全新的测试。通常，这种测试代价是很高的；②修改对象属性。

12.2.8　基于 UML 的面向对象软件测试

统一建模语言(Unified Modeling Language，UML)在 1997 年被国际标准化组织(OMG)接纳为正式官方标准后，已成为新一代面向对象软件设计的事实标准，其与 IBM Rational 统一过程(或其他软件工程开发过程)的配合使用更是被业界许多企业所采用。

UML提供了一套描述软件系统模型的概念和图形表示方法，以及语言的扩展机制和对象约束语言，软件开发人员可以使用UML对复杂的面向对象软件系统建立可视化的模型，并通过增量式的不断细化直接控制从设计、编码、测试到文档编制的整个软件开发过程。

UML 为使用者提供了丰富的图形表示方式，使得使用者可以从不同的抽象角度对软件系统的特征进行描述。UML 是一种图形化的设计语言，它使用不同类型的图，从不同的角度和抽象层次上描述系统模型。在使用 UML 进行软件设计建模时，设计者必须根据所需要描述的对象和系统动作选择适当的 UML 图，以达到设计效果。同样在软件测试过程中，在不同的测试阶段，根据不同的测试目的，必须选择不同的 UML 图。以下将分析几种常用 UML 图形的特点及其对软件测试的影响。

1. 类图

类图展现了一组对象、接口、协作和它们之间的关系。类图给出了系统的静态设计视图(包含主动类的类图)以及静态进程图。好的类图不仅能够描述系统中的类和这些类之间

的关系，而且可以针对方法层的类和对象引入附加的属性和关系。由于类图是对面向对象系统中类的最为详尽和全面的描述方式，因而能够帮助测试者全面掌握系统中的类结构，这对于类的测试有很大的帮助。

2. 用例图

用例图展现了一组用例、参与者以及它们之间的关系。它给出了系统的静态用例视图。用例图往往是在一个较高的抽象层次上描述系统的行为和各个组件之间的关系，所以能够帮助测试者全面把握系统的运行情况，是集成测试和系统测试的重要参考工具。

3. 状态图

状态图展现了一个状态机，它由状态、转换、事件和活动组成，是专注于系统的动态视图。由于状态图强调对象行为的事件顺序，因而它对于接口、类和协作的测试都有很重要的作用。由于建立在状态机的基础上，因此很多已有的针对状态机测试的有效方法都能很容易地被移植到对应状态图的测试中。通常是采用 W 方法构造所需的测试用例，或者将状态机转换为数据流图，依据数据流分析方法构造测试用例。

4. 序列图

序列图是一种强调消息的时间顺序的交互图，主要用来设计和描述算法。从测试的角度来看，序列图中可能存在一些错误，其中包括约定的冲突、不能创建正确的对象、图中没有关系的发送者和接收者之间的消息传递等。

在 9 种 UML 图中，把用于对系统的静态方面进行可视化、详述、构造和文档化的 UML 图称为结构图(参见表 12-2)。

表 12-2　结构图的名称和描述对象

名称	描述对象
类图	类、接口、协作
对象图	对象
构建图	构建
实施图	节点

同样，可以把用于对系统的动态方面进行可视化、详述、构造和文档化的 UML 图称为行为图(参见表 12-3)。

表 12-3　行为图的名称和描述对象

名称	描述对象
用例图	组织系统的行为
序列图	消息的时间顺序
协作图	收发消息的对象的结构组织
状态图	有事件驱动和系统的变化状况
活动图	从活动到活动的控制流

根据 UML 图的不同抽象层次，在软件开发和测试的不同阶段使用的 UML 图也各不相同(如表 12-4)。

表 12-4　UML 图与软件设计和测试的对应关系

软件设计	UML 图	
需求分析	软件测试用例图、实施图	确认测试
功能划分	活动图、构建图	系统测试
系统设计	状态图、对象图、协作图	集成测试
编码实现	状态图、对象图、协作图、序列图、类图	单元测试

12.3　面向对象软件测试用例的设计

传统软件测试用例的设计是从软件的各个模块的算法细节得出的，而面向对象的软件测试用例则着眼于适当的操作序列，以实现对类的说明。

黑盒测试不仅适用于传统软件，也适用于面向对象的软件测试。同样，白盒测试也适用于面向对象的软件类的操作定义，但面向对象的软件中许多类的操作结构简明，所以有人认为：在类层上测试可能要比传统软件中的白盒测试方便。

12.3.1　基于故障的测试

在面向对象的软件中，基于故障的测试具有较高的发现可能故障的能力。由于系统必须满足用户的需求，因此基于故障的测试要从分析模型开始，考查可能发生的故障。为了确定这些故障是否存在，可设计测试用例去执行设计或代码。

基于故障测试的关键取决于测试设计者如何理解可能的错误，而在实际中，要求设计者做到这点是不可能的。基于故障的测试也可以用于集成测试，集成测试可以发现消息联系中可能的故障。可能的故障一般为意料之外的结果，错误地使用了操作，消息不正确引用，等等。为了确定由操作(功能)引起的可能故障，必须检查操作的行为。这种方法除用于操作测试外，还可用于属性测试，用以确定其对于不同类型的对象行为是否赋予了正确的属性值(因为一个对象的属性是由赋予属性的值定义的)。

12.3.2　基于脚本的测试

基于脚本的测试主要关注用户需要做什么，而不是产品能做什么，即从用户任务(使用用例)中找出用户要做什么并去执行。这种基于脚本的测试有助于在单元测试情况下检查多重系统。所以基于脚本的测试比基于故障的测试不仅更实际(接近用户)，而且更复杂一点。

12.3.3　面向对象类的随机测试

如果一个类有多个操作(功能)，这些操作(功能)序列有多种排列，而这种不变化的操作序列可随机产生，用这种可随机排列的序列来检查不同类实例的生存史，就叫随机测试。

例如一个银行信用卡应用，其中有一个类：Account。Account 类的操作有：open、setup、deposit、withdraw、balance、sum-marize、creditlimit 和 close。

这些操作中的每一项都可用于计算，但 open 和 close 必须在其他计算的任何一个操作前后执行，即使 open 和 close 有这种限制，这些操作也仍有多种排列。所以一个不同变化的操作序列会因应用不同而随机产生，比如一个 Account 实例的最小行为生存史可包括以下操作：

```
open+setup+deposit+[deposit|withdraw|balance|summarize|creditlimit]+withdraw+close
```

由此可见，尽管这个操作序列是最小测试序列，但在这个序列内仍可以发生许多其他的行为。

1. 类层次的分割测试

这种测试可以减少用完全相同的方式检查类测试用例的数目。这很像传统软件测试中的等价类划分测试。分割测试又可分三种：

① 基于状态的分割，按类操作是否改变类的状态来分割(归类)。
② 基于属性的分割，按类操作用到的属性来分割(归类)。
③ 基于类型的分割，按完成的功能分割(归类)。

2. 由行为模型(状态图、活动图、顺序图和合作图)导出的测试状态转换图(Status Translation Diagram，STD)

可用来帮助导出类的动态行为的测试序列，以及这些类与合作类的动态行为测试序列。

为了说明问题，仍用前面讨论过的 Account 类。开始由 empty acct 状态转换为 setup acct 状态。类实例的大多数行为发生在 working acct 状态。而最后，取款和关闭分别使 Account 类转换到 non-working acct 和 dead acct 状态。这样，设计的测试用例应当完成所有的状态转换。换句话说，操作序列应当能导致 Account 类所有允许的状态进行转换。

测试用例：open+setup acct+deposit(initial)+withdraw(final)+close。还可导出更多的测试用例，以保证该类的所有行为被充分检查。

习题和思考题

1. 什么是面向对象技术？它和结构化程序设计方法相比有哪些特点？
2. 传统的软件测试策略是什么？为什么不适用于面向对象软件？
3. 为什么面向对象编程语言的有关语言特征会对软件测试产生影响？
4. 面向对象软件测试包括哪些内容？我们如何进行面向对象软件的测试？

参 考 文 献

[1] 蔡建平.软件测试大学教程.北京：清华大学出版社，2014
[2] 蔡建平.软件测试实验指导教程. 北京：清华大学出版社，2009
[3] 蔡建平.嵌入式软件测试实用技术. 北京：清华大学出版社，2010
[4] 百度百科：软件危机，http://baike.baidu.com/view/30093.htm
[5] 百度文库：软件开发前沿技术，https://wenku.baidu.com/view/718b4928910ef12d2af9e7e5.html，2016.6
[6] 百度文库：当今软件开发的主流技术评述，http://wenku.baidu.com/view/a06f2aea6294dd88d0d26bbb.html，2011.2
[7] 百度百科：面向服务架构，http://baike.baidu.com/view/453197.htm?fromId=6545280
[8] 百度文库：配置管理精髓，http://www.educity.cn/pm/625478.html，2013.12
[9] 百度文库：软件开发过程中的项目管理，http://wenku.baidu.com/view/a6b0f5dd5022aaea998f0f04.html，2011.2
[10] 陈文兵.基于软件测试的质量度量研究和应用.中国科学院大学，2015.01
[11] 邢薇薇、王新刚.航空机载软件缺陷分类方法研究及应用.测控技术，2016.9
[12] CSDN 博客：软件测试的生命周期&测试流程，http://blog.csdn.net/ChaosMax/article/details/71628867?locationNum=9&fps=1，2017.05
[13] 王诗琹，李彬，卢叶.基于 GJB5000A 的项目全生命周期软件测试实践与分析.通信技术，2016.11
[14] 百度文库：软件测试的原则，https://wenku.baidu.com/view/3036f8dcad51f01dc281f153.html?from=search，2012.4
[15] 百度文库：软件测试工作各测试阶段任务详解，http://wenku.baidu.com/view/a65b9d03b52acfc789ebc99a.html，2011.8
[16] 百度百科：软件测试员，http://baike.baidu.com/view/1984459.htm
[17] 道客巴巴：软件测试工作总体流程图，http://www.doc88.com/p-6711898312695.html，2015.01
[18] 马占军.软件测试认识误区之探讨.科技信息，2011 年第 34 期
[19] 百度文库：软件测试职业介绍，https://wenku.baidu.com/view/4c2f9977580216fc700afdf9.html，2014.05
[20] 豆丁：软件测试职业与能力介绍，http://www.docin.com/p-764073289.html，2014.2
[21] 百度文库：软件测试职业规划，https://wenku.baidu.com/view/4673e0ef10661ed9ac51f332.html?from=search，2016.2
[22] 北大青鸟：软件测试工程师应该具备哪些素质，http://www.sohu.com/a/119379321_516714，2016.11

[23] 健康网：软件测试人才就业前景看好，http://xl.99.com.cn/zcdt/210950.htm，2012.4
[24] CSDN 博客：软件测试的生命周期&测试流程，
http://blog.csdn.net/ChaosMax/article/details/71628867?locationNum=9&fps=1，2017.5
[25] 百度文库：软件测试的生命周期，
https://wenku.baidu.com/view/c79fcef3fab069dc502201ff.html，2013.2
[26] 51testing：测试生命周期，http://www.51testing.com/html/09/n-235309.html，2011.5
[27] CSDN：测试风险的管理，http://www.docin.com/p-1558217386.html，2016.5
[28] 百度文库：软件测试风险管理，
https://wenku.baidu.com/view/9ef3c69af121dd36a32d82f9.html?from=search，2013.9
[29] 刘志强.基于风险测试的软件测试方法研究.内蒙古民族大学学报，2012 年第 2 期
[30] Henry：测试管理实践，http://www.docin.com/p-589412070.html，2013.1
[31] 百度文库，软件测试过程，
http://wenku.baidu.com/view/7f68bbc058f5f61fb736668c.html，2012.9
[32] 百度文库：软件测试基本流程与规范，
https://wenku.baidu.com/view/d6887d8db9f3f90f76c61be5.html，2014.8
[33] 百度文库：测试成熟度模型集成(TMMi)中文，
https://wenku.baidu.com/view/44ecc52b7cd184254a353525.html?from=search，2013.9
[34] 51Testing 软件测试网：软件测试之需求分析，
http://www.51testing.com/html/36/n-3704236.html，2016.2
[35] 百度文库：测试需求分析，
http://wenku.baidu.com/view/e6ee231055270722192ef7f6.html，2012.11
[36] 软件测试网：浅谈测试需求分析，
http://www.51testing.com/html/92/n-139492.html，2012.11
[37] 互动百科：测试用例，
http://www.baike.com/wiki/%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B
[38] 工程硕士学位论文：软件测试用例生成及管理系统的设计和实现，吉林大学，2012.12
[39] 百度文库：如何组建管理测试团队，
https://wenku.baidu.com/view/c16b197401f69e3143329477.html?from=search，2012.4
[40] 豆丁：软件测试团队组织及任务分配，
http://www.docin.com/p-222109360.html#documentinfo
[41] GJB/Z 141-2004 军用软件测试指南
[42] 百度文库：国标或军标中对测试级别和测试类型的规定，
http://wenku.baidu.com/view/b643bdfffab069dc50220137.html，2012.4
[43] UML 软件工程组织：软件测试项目管理系之测试类型划分及其项目实施，
http://www.uml.org.cn/Test/201211135.asp，2012.11
[44] CSDN：信产公司软件测试等级划分，
http://download.csdn.net/download/kingsang_ll/4722074，2012.11

[45] 博客：软件测试通过及 BUG 分级标准，
http://blog.csdn.net/csq2002/article/details/38023367，2014.7
[46] 百度文库：软件测试种类，
https://wenku.baidu.com/view/19bf8e73bb68a98271fefab8.html，2015.3
[47] 百度文库：静态测试，http://wenku.baidu.com/view/ef29f6fcfab069dc50220148.html，2012.6
[48] 百度文库：静态测试技术，
https://wenku.baidu.com/view/d71b760aa417866fb84a8eb0.html?from=search，2015.4
[49] 李龙.软件测试实用技术与常用模板.北京：机械工业出版社，2010 年 10 月
[50] 百度文库：Code Review 理论与实战，
http://wenku.baidu.com/view/59bb9e2bb4daa58da0114a98.html，2012.1
[51] andy572633：白盒测试方法—代码检查法，
http://blog.csdn.net/andy572633/article/details/7266277，2012.2
[52] andy572633：白盒测试方法—静态结构分析法，
http://blog.csdn.net/andy572633/article/details/7269484，2012.2
[53] andy572633：白盒测试方法—静态质量度量法，
http://blog.csdn.net/andy572633/article/details/7273214，2012.2
[54] 百度文库：软件的复杂性度量方法概述，
http://wenku.baidu.com/view/fcee483c87c24028915fc377.html，2012.6
[55] 曾一 等.面向对象类的复杂性的度量方法.计算机工程与应用，2010，46(12)
[56] 百度文库：软件度量综述，
https://wenku.baidu.com/view/7f338e945f0e7cd1842536a3.html?from=search，2015.5
[57] 百度文库：国标质量特性测试方法建议指导书，
http://wenku.baidu.com/view/919e3654ad02de80d4d840dc.html，2012.1
[58] 百度文库：白盒测试的方法，
https://wenku.baidu.com/view/bc85c679eefdc8d376ee32e9.html?from=search，2014.5
[59] 软件测试网：白盒测试方法，
http://www.51testing.com/ddimg/uploadsoft/2007/baiheceshifangfa.pdf
[60] 百度文库：数据流测试，
https://wenku.baidu.com/view/b939b3b987c24028905fc345.html?from=search，2015.5
[61] 百度文库：软件结构性测试，
https://wenku.baidu.com/view/9440f01a8e9951e79b8927a5.html?from=search，2014.12
[62] 百度文库：软件白盒测试，
https://wenku.baidu.com/view/37912e8027d3240c8547efcf.html?from=search，2016.3
[63] 百度文库：基本路径测试，
https://wenku.baidu.com/view/ec39cdbed0d233d4b14e69e6.html?from=search，2013.12
[64] 百度文库：测试覆盖率，
https://wenku.baidu.com/view/165dd31155270722192ef75a.html?from=search，2012.4
[65] 百度文库：黑盒测试，
http://wenku.baidu.com/view/e32d9c926bec0975f465e272.html，2012.8

[66] 百度文库：测试用例设计—黑盒测试，
http://wenku.baidu.com/view/75372e33a32d7375a4178019.html，2012.4
[67] 百度文库：如何编写有效测试用例，
https://wenku.baidu.com/view/12684750482fb4daa58d4ba3.html?from=search，2015.1
[68] CSDN 博客：如何划分测试用例的优先级别，
http://blog.csdn.net/wuyanhong1985/article/details/8171738，2012.11
[69] 博客园：测试用例八大要素，http://www.cnblogs.com/qjm520/p/5832109.html，2016.9
[70] 百度文库：测试用例标准，
https://wenku.baidu.com/view/e12fb3edd4d8d15abe234e9f.html?from=search，2014.12
[71] 百度文库：软件测试基本理论和方法，
https://wenku.baidu.com/view/99fd0afabceb19e8b8f6baa2.html?from=search，2015.4
[72] 百度文库：软件单元测试技巧，
http://wenku.baidu.com/view/1d15bc275901020207409cd9.html，2012.3
[73] 百度文库：单元测试规范，
https://wenku.baidu.com/view/9765286f6bec0975f565e227.html?from=search，2016.10
[74] 百度百科：集成测试，http://baike.baidu.com/view/106652.htm
[75] 百度文库：实用软件测试方法与应用(集成测试)，
http://wenku.baidu.com/view/dda7dc8f8762caaedd33d43c.html，2013.1
[76] 百度文库：软件测试理论，
https://wenku.baidu.com/view/06d608fa90c69ec3d4bb756e.html?from=search，2016.10
[77] 百度文库：系统测试过程，
https://wenku.baidu.com/view/77763b73168884868762d67c.html?from=search，2013.3
[78] 童战.软件易用性测试研究，http://www.uml.org.cn/Test/201209213.asp，2012.9